Sistemas Informáticos
Curso 2013-2014

Facultad de Informática
Universidad Complutense de Madrid

SISTEMAS OPEN DATA

Aplicaciones de Medio Ambiente

ÁLVARO HERNÁNDO GAVILÁN

ANTONIO IRÍZAR LÓPEZ

CARLOS RODRÍGUEZ DÍAZ

Dirigido por:

Dra. Victoria López López

Dra. Inmaculada Pardines Lence

Autorización de Difusión

Álvaro Hernando Gavilán, Antonio Irízar López y Carlos Rodríguez Díaz, alumnos matriculados en la asignatura Sistemas Informáticos, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria como el código, la documentación y/o el prototipo desarrollado en el proyecto Sistemas Open Data – Aplicaciones de Medio Ambiente, todo ello realizado durante el curso académico 2013-2014 bajo la dirección de Victoria López López e Inmaculada Pardines Lence, ambas profesoras del Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática.

Álvaro Hernando Gavilán Antonio Irízar López Carlos Rodríguez Díaz

Resumen

Sistemas Open Data es un proyecto que estudia las distintas posibilidades que existen para el tratamiento de datos de carácter abierto. Concretamente, se centra en las iniciativas *Open Government* o de Gobierno Abierto, que están enfocadas en aquellos datos que se crean y gestionan en las organizaciones gubernamentales de los países.

Entre las aspiraciones que tienen dichas iniciativas, está la de motivar ejercicios de transparencia entre los gobiernos y la ciudadanía, de manera que mejoren la calidad de las democracias actuales. Por otra parte, la publicación y el libre acceso a los datos posibilita la generación de riqueza y valor a partir de ellos, lo que ha derivado en el fenómeno de las *Smart Cities* o Ciudades Inteligentes, las cuales son poblaciones dotadas de multitud de aplicaciones desarrolladas en base a los datos que las administraciones públicas divulgan.

Durante los años 2012-2013, se implementaron en la Facultad de Informática unas aplicaciones móviles para *Android* utilizando datos del Ayuntamiento de Madrid (las **Aplicaciones de Medio Ambiente** del grupo G-Tec de la UCM). Estas apps obtienen sus datos fuente desde servidores independientes y destinados a ser usados únicamente por dichas aplicaciones. Resulta natural pensar que, si las aplicaciones usan información proporcionada por una institución estatal como es el Ayuntamiento, esta se encuentre disponible de manera pública, libre y agrupada en un mismo sitio (un único servidor).

Así surgió el objetivo principal de este proyecto: aplicar los principios analizados relativos a *Open Data* y *Open Government* sobre las aplicaciones de Medio Ambiente. Para hacer realidad este deseo, hemos propuesto un sistema Open Data para el Ayuntamiento de Madrid, que además, sirva de soporte tanto para las aplicaciones de G-Tec, como para las que se desarrollen en un futuro. Para demostrar esto último, hemos adaptado una de las apps, así como desarrollado una página web con funcionalidad equivalente a dicha aplicación. Con ello pretendemos probar que estos sistemas no solo están orientados a aplicaciones móviles, sino que están abiertos a infinidad de propuestas que conviertan a Madrid en una Ciudad Inteligente, y además, en un ejemplo de transparencia.

Palabras Clave

Open Data, Datos Abiertos, Android, Aplicaciones móviles, CKAN, Amazon Web Service, bases de datos, conjunto de datos, dataset, Madrid, Medio Ambiente, Google Maps, mapas, Ngnix, PostgreSQL, Solr, Apache, Servidores, VirtualBox, Virtualización, Servicios en la nube, Python, Framework Pylons, API, REST, SQLAlchemy, WSGI, Proxy, hipervisor, PHP, Java, máquina virtual, Ubuntu, Prolog, CSV, XML.

Abstract

Sistemas Open Data (Open Data Systems) is a project that study the possibilities for open data treatment. Specifically, it focuses on Open Government initiatives, which are focused on data that is created and managed in government organizations.

Among the aspirations of those initiatives, it exists the fact to motivate transparency practises between governments and citizens, so as to improve the quality of the current democracies. Moreover, the publication of this data and its free availability, enables the creation of wealth and generates new values from them, which has led to the phenomenon of the Smart Cities. These populations have developed new services (websites, mobile apps, etc.) that use the data that its government's Open Data hosts.

During the years 2012-2013, several Android mobile apps (G-Tec UCM Enviroment Apps) which use data from the city hall of Madrid were implemented at Computer Science Faculty of University Complutense of Madrid. These apps get their source data from separate servers that are intended for use only by those applications. It is natural to think that if these apps use the data provided by a state institution as the City Council is, these data should be free, publicly available, and clustered in one place (a single server).

This idea derived into the main aim of this project: applying the discussed principles related to Open Data and Open Government, on the apps of Environment that were mentioned before. To make this wish come true, we have proposed to develop an Open Data system for the city of Madrid, which will serve as support for the G-Tec Apps and future new apps. To prove the latter, we have adapted one of these apps and we've developed its web version. Our goal is to prove that these systems are not only focused on mobile applications, they are also open to countless proposals that will get Madrid to become a Smart City, and also an example of transparency.

Keywords

Open Data, Android, mobile apps, CKAN, Amazon Web Service, Madrid, databases, Medio Ambiente, dataset, Google Maps, maps, Ngnix, PostgreSQL, Solr, Apache, Servidores, VirtualBox, Virtualization, Cloud Services, Python, Framework Pylons, API, REST, SQLAlchemy, WSGI, Proxy, hypervisor, PHP, Java, máquina virtual, Ubuntu, Prolog, CSV, XML.

Dedicado a nuestras familias y amigos.

Gracias.

Agradecimientos

Nos gustaría agradecer a Victoria López López e Inmaculada Pardines Lence, que han sido nuestras tutoras a lo largo del desarrollo de este proyecto, los consejos y la ayuda que nos han prestado para concluir con éxito este estudio. También queríamos hacer una mención especial a Óscar Sánchez por la ayuda facilitada, así como al personal del Ayuntamiento por el tiempo y paciencia que nos han dedicado.

Por último, nos gustaría agradecer a nuestras familias, amigos y compañeros su apoyo durante estos últimos años.

Gracias a todos.

Índice de contenidos

I. Introducción

Desde tiempos remotos, el ser humano se ha visto incitado y motivado a almacenar y gestionar todo tipo de información. Ya por el siglo III a.C, Ptolomeo I Soter fundó la conocida Biblioteca de Alejandría, la cual albergaba hasta 900.000 manuscritos con contenidos de todo tipo de disciplinas. A día de hoy la realidad es bastante distinta. La humanidad ya no concentra toda la información únicamente en las bibliotecas o en las universidades. En la actualidad, todo tipo de instituciones, entidades, empresas, e incluso los propios Estados, reúnen diferentes conjuntos de información.

Hasta no hace mucho tiempo, la información se guardaba y clasificaba en grandes salas repletas de archivos y ficheros. De esta manera, el acceso a los datos se debía hacer de forma manual. Sin embargo, esta solución no es factible cuando el volumen de la información comienza a crecer sin control como puede ocurrir, por ejemplo, en una entidad financiera.

La revolución tecnológica ha supuesto un gran alivio para este problema. Se han sustituido los tediosos ficheros de papel por máquinas y computadores que, junto a herramientas como los sistemas de gestión de bases de datos, han permitido una manipulación e indexación más ágil de la información. Por otra parte, esta mejora también se vio reflejada en la capacidad de almacenamiento física de la información, pues lo que antes ocupaba un piso entero de un edificio, ahora es posible guardarlo en un simple disco duro; o dicho de otro modo, se pueden acumular más datos y acceder a ellos más rápido.

Este último hecho ha dado lugar a un crecimiento exponencial en el volumen de datos que se manejan hoy en día en el mundo, es más, el 90% de esos datos se han creado tan solo desde 2010 [1] y parece que esta tendencia seguirá en aumento. Todo esto surge como resultado de que en la actualidad existe un número de dispositivos (ordenadores, *tablets, smartphones*, sensores…) conectados a la red equivalente a la población mundial y se espera que para 2015 este número se haya duplicado.

Todo este crecimiento tanto en avances tecnológicos como en volumen de datos está provocando cambios en las metodologías de toma de decisiones y abriendo nuevos enfoques de pensamiento. Así se llega hasta el desafío actual: **Obtener el valor intrínseco que los propios datos encierran**, es decir, aprovechar la alta cantidad de datos recolectada para obtener resultados que a simple vista no serían observables.

Esta idea se puede ilustrar con un ejemplo sencillo: Supongamos una cadena de hipermercados, que además de mantener una referencia y el importe de una factura que emitió en algún momento, también almacena la lista de artículos que cada cliente adquirió. Esta información adicional, debidamente tratada y procesada, permite la detección de comportamientos y tendencias de compra entre los clientes.

Pero manejar y analizar estas enormes cantidades de datos (estructurados, no estructurados y semi-estructurados) mediante una base de datos relacional conlleva costes demasiado altos (sobre todo en tiempo). Esto nos conduce a la necesidad de un nuevo concepto conocido como Big Data y a unos nuevos modelos de negocio que nos planteamos a partir de cómo utilizar la información que está a nuestro alcance.

1.2 Big Data

1.2.1 Definición

Big Data es el término que se utiliza para describir el conjunto de procesos, tecnologías y modelos de negocio que se basan en datos, y en explorar y explotar el valor que los propios datos poseen [1]. Esta tarea se puede llevar a cabo desde una visión más tradicional (mejorando la eficiencia en el análisis al poseer más información) o tratar de buscar nuevas formas de usar los datos almacenados de manera que se obtenga nueva información que antes se ignoraba (como en el ejemplo anterior de las facturas). Encontrar nuevos valores para los datos permite crear nuevos modelos de negocio, por ello, esta segunda visión es la que más interés está cobrando.

El Big Data se caracteriza por las *tres uves:*

- **Volumen:** Hasta hace unos años, los datos de una empresa o institución ocupaban unos cuantos MegaBytes o GigaBytes, pero con el auge de los mercados online y los nuevos espacios de negocio, actualmente las empresas han pasado a manejar volúmenes de datos del orden de PetaBytes. Para hacernos una idea, en un solo día Google procesa unos 20 PetaBytes, la Bolsa de Nueva York genera 1 terabyte, Twitter crea unos 80 MB nuevos cada segundo, etc.

- **Variedad:** Hoy en día se trabaja con una amplia gama de datos, tanto estructurados (almacenados en bases de datos) como desestructurados o incluso a medio camino entre ambos, como es el caso de los formatos de video y audio, el XML, etc. Las fuentes generadoras de información también son variadas y van desde los servicios de audio y video en *streaming* hasta los datos bancarios y de cotizaciones bursátiles, datos de sensores, etc. Todos estos conjuntos de datos precisan un análisis para convertirlos en información útil.

- **Velocidad:** Es de esperar que la velocidad de respuesta de los sistemas sea lo suficientemente rápida para alcanzar resultados correctos en el instante preciso. Es el caso de los sensores digitales en equipos industriales, GPS, medidores electrónicos, etc., donde esta característica es crítica, dado que se necesita

procesar mucha cantidad de información y dar una respuesta consecuente en tiempo real.

1.2.2 Fuentes y tipos de datos

Podemos clasificar los datos según provengan de fuentes externas o internas o sean de tipo estructurado o desestructurado:

- Fuentes **externas:**
    - **Estructurados:** Se obtienen del análisis de los datos internos de una empresa o entidad, como los datos del censo, historiales de crédito o viaje, etc.
    - **Desestructurados:** Son los datos que usan las entidades para definir o establecer comportamientos de consumo de los clientes. Esto se hace a través de las redes sociales o blogs que se han convertido en los nuevos medidores de opinión y en ellos es donde se pueden recoger críticas o promocionar los servicios al disponer de un canal directo con los clientes. Estos son los datos en los que las empresas y entidades están poniendo mayor atención.
- Fuentes **internas:**
    - **Estructurados:** Son los datos que tradicionalmente han venido manejando las empresas. Son sus datos financieros, registros de ventas, de recursos humanos, etc.
    - **Desestructurados:** Sirven de escenario de aprendizaje a las entidades para entender y familiarizarse con los datos desestructurados de manera que luego sean capaces de manejar los datos desestructurados externos. Aquí se sitúan los foros online, los *feeds* de las webs, documentos de texto, etc.

1.2.3 Oportunidades

Desde el punto de vista de los negocios, la filosofía de trabajo está cambiando del *"data science*" (ciencia de datos) o método tradicional al *"data intelligence"* (inteligencia de datos) en el que se intenta sacar partido al valor oculto de la información. Desde el punto de vista de las tendencias, el siguiente paso es encontrar el valor empresarial y de negocio al utilizar Big Data, es decir, de qué manera se puede beneficiar un negocio al utilizar Big Data o que nuevos modelos de negocio se pueden generar a partir de los datos obtenidos.

Hasta la fecha, las empresas y entidades que decidan incorporar un Big Data pueden ya aprovechar algunas ventajas en diferentes campos como el conocimiento más fino del cliente, marketing, gestión del riesgo, etc., Algunos de estos ejemplos son:

- **Análisis de los usuarios o clientes:** Existe la posibilidad de monitorizar sus pautas para poder ofrecerles productos, servicios, recomendadores o marketing totalmente adaptado a sus preferencias.

- **Análisis de marketing:** Optimización de modelos de marketing y promociones para evaluar y obtener un seguimiento su rentabilidad. Es posible también usarlo para regular los precios al evaluar el impacto de los mismos frente a la demanda de productos.

- **Efectividad operativa:** Poseer una gran cantidad de datos permite analizarlos en busca de mejoras de la planificación, calidad, recursos, etc., de manera que redunde en una mejora de la productividad.

- **Análisis de fraude:** Análisis de los datos de los clientes para descubrir conductas fuera de lo común, detectar el fraude y los riesgos. También se puede realizar un seguimiento completo de las transacciones en tiempo real gracias a los datos disponibles en los puntos de venta.

- **Análisis de internet/redes sociales/medios:** Como dijimos anteriormente, las opiniones en blogs y redes sociales y su impacto en la sociedad están suponiendo una nueva métrica de obligado uso para la evaluación de los productos.

1.2.4 Modelos de negocio

Podemos agrupar en tres grupos los nuevos planteamientos basados en la información:

- **Diferenciación:** Se trata de satisfacer las necesidades del cliente a través de un trato personalizado ajustado a sus preferencias y diferenciado para cada usuario.

- **Intercambio:** Analizar la información para posteriormente poner a la venta los resultados o conclusiones de la misma.

- **Redes de distribución:** Facilitar el intercambio de información y su publicidad fomentando nuevos mercados de información.

1.3 Open Data

Open Data [2] (en español, *Datos Abiertos*) es un concepto que surge de la idea de que ciertos datos deberían estar gratuitamente disponibles a todo aquel que los necesite para su uso y publicación de la forma que se desee, sin restricciones por copyright, patentes u otros mecanismos de control. El objetivo del movimiento Open Data es similar al de otros movimientos "*Open*" tales como *open source*, *open hardware*, *open content* y *open access*. La filosofía detrás de Open Data se ha ido estableciendo a lo largo del tiempo, pero el término Open Data en si es más actual, ganando popularidad con el surgimiento de Internet y *World Wide Web* pero, sobre todo, con el más reciente lanzamiento de iniciativas de Gobierno Open Data tales como data.gov [3] (EE. UU.) y data.gov.uk [4] (Reino Unido).

Así pues, una definición formal del concepto "Open Data" es algo relativamente nuevo, y vendría a decir que "*Unos datos son abiertos si cualquier persona tiene libertad para usarlos, reutilizarlos y redistribuirlos*".

Según el manual de Open Data [*Open Knowledge Foundation*, 2012], para que un sistema sea declarado como Open Data, debe cumplir las siguientes características fundamentales

El acceso y la disponibilidad de los datos deberá ser preferentemente por Internet de manera que el alcance sea el mayor posible.

Los datos se podrán reutilizar y distribuir de forma totalmente libre dando pie a la creación de nuevos servicios que aumenten el valor social y comercial de la sociedad.

Por lo tanto, un Open Data es un caso particular de un Big Data en el que los datos son de dominio público.

El movimiento de datos abiertos está a menudo enfocado en material no textual como mapas, fórmulas científicas y matemáticas, datos y prácticas médicas, etc. Es común que surjan problemas debido a que dichos datos tienen o pueden tener un valor comercial, por lo que su acceso o reutilización es controlada por organizaciones privadas o públicas. Para ello se utilizan restricciones de acceso, licencias, copyright, patentes y cargos por acceso o uso. Los defensores de Open Data argumentan que estas restricciones van en contra del bien común y que todos esos datos deberían estar disponibles sin restricciones o tasas.

Una inteligente cita que manifiesta la necesidad de Open Data sería:

"*Numerosos científicos han señalado la ironía de que precisamente en el momento histórico en el que disponemos de las tecnologías que permiten la disponibilidad a nivel mundial y el proceso de distribución de datos científicos, ampliando la colaboración y acelerando el ritmo y profundidad del descubrimiento... Estamos preocupados aislando y*

ocultando esos datos y evitando el uso de dicha tecnología avanzada en conocimiento." John Wilbanks, VP Science, Creative Commons.

Los creadores de datos a menudo no consideran la necesidad de establecer las condiciones de propiedad y las concesiones de licencias o reutilización. Por ejemplo, muchos científicos no consideran bajo su control los datos que se derivan de su trabajo, y su publicación en un periódico es una liberación implícita de dichos datos en el medio común. Sin embargo, la falta de una licencia hace difícil determinar el estado de un conjunto de datos y sus restricciones en cuanto al uso de dichos datos ofrecidos en un espíritu abierto. Además, debido a esta incertidumbre, es posible para organizaciones públicas o privadas reunir esos datos, protegerlos con copyright y entonces revenderlos.

Open Data puede provenir de diferentes fuentes, si bien las principales son la ciencia, y el gobierno, como es el caso de nuestro proyecto.

Detrás del *Open Data Government* hay dos razones fundamentales. Primero, defensores del movimiento sostienen que la disponibilidad en formatos abiertos aumenta la transparencia y responsabilidad del gobierno. En segundo lugar, Open Data debería permitir a terceros aprovechar el potencial de datos del gobierno en el desarrollo de aplicaciones y servicios que genere demanda pública y privada.

Varios gobiernos nacionales han creado sitios web para distribuir parte de los datos que recogen. Se trata de un proyecto de colaboración con gobiernos municipales para crear y organizar cultura de Open Data u Open Government Data. Alrededor de 200 catálogos (locales, regionales y nacionales) de datos abiertos se encuentran reunidos en el proyecto "open source" datacatalogs.org, el cual se puede considerar una lista de catálogos de datos de alrededor de todo el mundo. Buenos ejemplos, como ya señalamos anteriormente, son los Open Data de los gobiernos de Estados Unidos y Reino Unido, data.gov y data.gov.uk respectivamente, ambos inaugurados en 2009.

1.4 Open Government

Open Government [5] (en español, *Gobierno Abierto*) es una doctrina que sostiene que los ciudadanos tienen derecho a acceder a los documentos y procedimientos del gobierno para permitir una supervisión pública efectiva [6]. En su sentido más amplio, se opone a la razón de Estado y otras consideraciones, que tienen tendencia a legitimar un extenso secreto de Estado. Los orígenes sobre el concepto de Open Government pueden ser datados a la época de la Ilustración en Europa: a los debates sobre la adecuada construcción de una entonces naciente sociedad democrática

Entre los acontecimientos más recientes está la teoría de gestión de código abierto, que aboga por la aplicación del movimiento de software libre a los principios democráticos, lo que permite a los ciudadanos interesados involucrarse más directamente en el proceso legislativo.

El concepto de Open Government se sustenta en tres pilares fundamentales:

- **Transparencia:** El gobierno y sus administraciones facilitarán a los ciudadanos el acceso a la información que maneja y a los planes de actuación que tiene de una forma sencilla y libre, de manera que se pueda supervisar al gobierno.

- **Colaboración:** El gobierno ha de colaborar con los ciudadanos, empresas, asociaciones, etc.

- **Participación:** Los ciudadanos pueden participar activamente en las decisiones que toma el gobierno, el cual se puede beneficiar de los conocimientos de la ciudadanía.

Estas definiciones se basan en las presentes en el Memorando sobre Transparencia y Gobierno Abierto [7] que escribió Barack Obama en contribución a la Alianza para el Gobierno Abierto, de la que España es integrante desde septiembre de 2011. Los principios de Transparencia, Colaboración y Participación derivaron en diversas estrategias de Open Government, pero siempre con dos ideas principales:

Disponibilidad pública de los datos que las administraciones poseen mediante estructuras Open Data que concedan libre acceso y reutilización de los datos. Estos datos pueden ser censos, registros, mapas, etc. De esta manera, los ciudadanos podrían generar valor a partir de ellos. Es algo natural que se liberen estos datos, dado que las administraciones los producen haciendo uso del dinero de los ciudadanos recaudado con los impuestos. Esto se conoce como Reutilización de la Información del Sector Público (RISP).

Utilización de las redes sociales y diferentes tipos de plataformas que permitan la comunicación y participación de los ciudadanos en el gobierno. No solo como un nuevo canal de contacto desde el gobierno a la ciudadanía, sino como herramientas que permitan la escucha activa de la opinión de los habitantes, incrementando la participación de estos en las democracias actuales. Esto es lo que se conoce como *"Open Action"*.

II. Estado del Arte

En esta sección vamos a exponer la importancia de seguir la filosofía de datos abiertos en la sociedad actual, haciendo especial hincapié en la utilización de sistemas Open Data, mediante el ejemplo de iniciativas "open" que se han venido desarrollando en los últimos años por todo el mundo.

Nos centraremos en el ámbito "gobierno abierto", pues el surgimiento de conceptos como "ciudad inteligente" (Smart City) fueron los que llevaron a los gobiernos a comenzar proyectos para dejar a disposición del público los datos de los que dispone, en formatos estructurados y a través de Internet. Y qué mejor manera de hacer esto que desarrollar un sistema Open Data.

2.1 Smart City

Actualmente, uno de los principales objetivos de los países desarrollados, y muy especialmente en Europa, es encontrar la manera de diseñar y adaptar sus ciudades para convertirlas en entornos inteligentes y sostenibles. Prácticamente las tres cuartas partes del los habitantes del continente europeo viven en ciudades, las cuales consumen el 70% de toda la energía de la Unión Europea. Los atascos producidos por el tráfico en Europa suponen anualmente un coste del 1% de su Producto Interior Bruto (PIB), y la mayoría de dichos atascos tienen lugar en las ciudades [8].

"*Smart City*" (en castellano, Ciudad Inteligente) se hace cada día más popular a nivel internacional, de la mano de términos como Big Data, Open Data u Open Government, y es que todos están fuertemente relacionados entre sí, en su acometida por la utilización de las NTIC (nuevas tecnologías de la información y la comunicación) para conseguir un gran desarrollo e innovación en diferentes ámbitos del mundo en el que vivimos: la ciencia, la información de cualquier tipo, su intercambio y su disponibilidad, el urbanismo, el gobierno, etc.

Aunque "Smart City" [9] también es un concepto emergente que se está utilizando mucho en marketing (mercadotecnia). Formalmente podríamos decir que una ciudad es "inteligente", cuando su desarrollo urbano está basado en la sostenibilidad, y la inversión social, el capital humano, las comunicaciones, y las infraestructuras, conviven de forma armónica con un desarrollo económico sostenible, apoyándose en la utilización y la modernización de las nuevas tecnologías (NTIC), y ofreciendo una mejor calidad de vida y una gestión prudente de los recursos naturales, a través de la participación y el compromiso de todos los ciudadanos.

Desde el punto de vista tecnológico, una "Smart City" es un sistema ecosostenible muy complejo y que contiene a su vez muchos subsistemas, es decir, un ecosistema global en el que coexisten múltiples procesos estrechamente ligados y que resulta difícil de abordar y valorar de forma individual.

En definitiva, el objetivo principal es conseguir un desarrollo urbano o territorial que mejore activamente la calidad de vida de las personas, satisfaciendo las necesidades tanto de las empresas e instituciones como de los ciudadanos, debido al uso ampliado de las NTICs.

Sin embargo, la categoría *"Smart*" no debe ser entendida como una meta determinada para la ciudad, sino como un compromiso por parte de los distintos agentes involucrados, de alcanzar un proceso de mejora constante, con un potencial prácticamente infinito, y una meta no menos lejana.

"Ser Smart City no es un objetivo en sí mismo. Es un medio para un fin. (… y el camino por el que una ciudad debe) seguir avanzando para ser, cada día más, sinónimo de oportunidades, cohesión y calidad de vida." – Alcaldesa de Madrid Ana Botella, página 3 de su discurso del día 7 de mayo de 2013.

A continuación podemos observar un listado con los principales ejemplos de proyectos "Smart City" que se están desarrollando actualmente en diferentes ciudades de todo el mundo:

- En Lyon, Francia, han desarrollado una "ciudad inteligente" estratégica para reforzar su política de desarrollo económico destinado a empresas de cualquier tamaño.

- Proyecto "Smart City" en Vigo, dedicado principalmente al reforzamiento de la comunicación, de la coordinación, y de la integración.

- Masdar es una "ecociudad inteligente" que se encuentra en construcción en el desierto de Abou Dabi.

- "*Amsterdam Smart City"* tiene como principal proyecto la creación de "AMS" (*Amsterdam Metropolitan Solution*), es decir, "Soluciones para el área metropolitana de Ámsterdam" y creación en dicha ciudad de un instituto de tecnología aplicada.

- En Egipto podemos encontrar el proyecto *"Cairo Smart Village"*.

- El Gobierno de Dubái, con sus proyectos "Dubái Smart City" y "Dubái Internet City", ha creado un parque tecnológico como zona franca y una base estratégica para compañías que apunten a mercados emergentes locales.

- *SmartSantander*, probablemente la mayor red de sensores del mundo en la actualidad. Este proyecto llevó a cabo la instalación de, por el momento, 1100 captores sin hilos de la sociedad Libelium, 400 de ellos para medir los lugares de aparcamiento, y 700 para controlar parámetros ambientales tales como el ruido, el monóxido de carbono, la temperatura o la luz solar.

- En Yokohama, una de las ciudades más grandes de Japón, podemos observar el *"Yokohama Smart City Project"* (YSCP), que proyecta construir una infraestructura de generación energética. Entre otros, pretende reducir las emisiones de dióxido de carbono y ser vanguardista en cuanto al sistema social y la protección de la naturaleza.

- *Open Cities* es un proyecto co-fundado por la Unión Europea que tiene como objetivo validar la forma de abordar metodologías de innovación *"Open"* y *"User Drive"* del sector público en un escenario de Servicios de Internet Futuros para *"Smart Cities"*. Se está desarrollando aprovechando herramientas, pruebas y plataformas existentes en *"Crowdsourcing", "Open Data", "Fiber to the Home"* y *"Open Sensor Networks"* en siete grandes ciudades europeas: Helsinki, Berlín, Ámsterdam, París, Roma, Barcelona y Bolonia.

Y es que las tecnologías que sugiere una "Smart City" pueden contribuir de forma muy significativa en la solución de muchos de los desafíos de una ciudad. Es más, la adopción de estrategias de apertura y reutilización de datos públicos (Open Data) y de gobierno abierto (Open Government), permiten una mayor participación por parte de cualquier ciudadano en el diseño y la puesta en marcha de servicios, dando a los habitantes de una ciudad el poder para tomar decisiones más inteligentes y "verdes" en su vida diaria. Además, hacen más transparentes y dignos de confianza al Gobierno y a las entidades públicas que administran nuestras ciudades, y posibilitan la participación de las empresas y de los ciudadanos en un proceso de diálogo continuo.

Como podemos observar, la creación de sistemas Open Data es un elemento clave de la filosofía "Smart City", y es que, como ya hemos planteado en la Introducción, la publicación abierta y estructurada de diversos datos de los que dispone el Gobierno, otras entidades o cualquier persona, es hoy día muy asequible debido a las nuevas tecnologías de la información y la comunicación, y ofrece desde ya la creación de nuevos valores a partir de dichos datos, dando a las personas la posibilidad de crear nuevos servicios que generen un desarrollo sostenible constante en el que no hay límites, sino imaginación.

2.2 Open Data en la actualidad

Hoy día, como ya comentamos en la introducción, existe un gran número de iniciativas "open" a lo largo de todo el mundo, especialmente en los dos campos más centrados en estas nuevas metodologías: ciencia y política. Este último es el más popular actualmente entre los Open Data, pues prácticamente todas las naciones desarrolladas del mundo entero ya tienen (o han comenzado su desarrollo) un portal de datos abiertos que ofrece la posibilidad de crear nuevos servicios y aplicaciones, y permite la participación de los ciudadanos para incrementar el desarrollo tecnológico y económico común del gobierno y sus habitantes.

A continuación vamos a exponer algunos de los Open Data que se han desarrollado en la geografía española así como alguno muy relevante a nivel mundial. Estos Open Data nos han servido como punto de partida para la tarea de desarrollar nuestra propuesta de Open Data para el Ayuntamiento de Madrid.

2.2.1 Open Data Extranjeros

En esta sección vamos a describir alguno de los Open Data extranjeros más interesantes.

❋ Data.gov

El *website* data.gov es el Open Data del gobierno de los Estados Unidos [10], publicado a finales de Mayo de 2009, por lo que se trata de uno de los primeros sistemas de datos abiertos desarrollado con la filosofía "Open", si no el primero en la categoría de "gobierno abierto".

De acuerdo con su web, *"el propósito de data.gov es incrementar el acceso público a datos de alto valor, conjuntos de datos legibles por una máquina generados por el Poder Ejecutivo del Gobierno Federal"*.

El sitio, siguiendo la filosofía de gobierno abierto (transparencia, colaboración, participación), pretende convertirse en un repositorio de toda la información que el gobierno recopila, dejando a disponibilidad del público todos los datos que no sean privados o restringidos por razones de seguridad nacional.

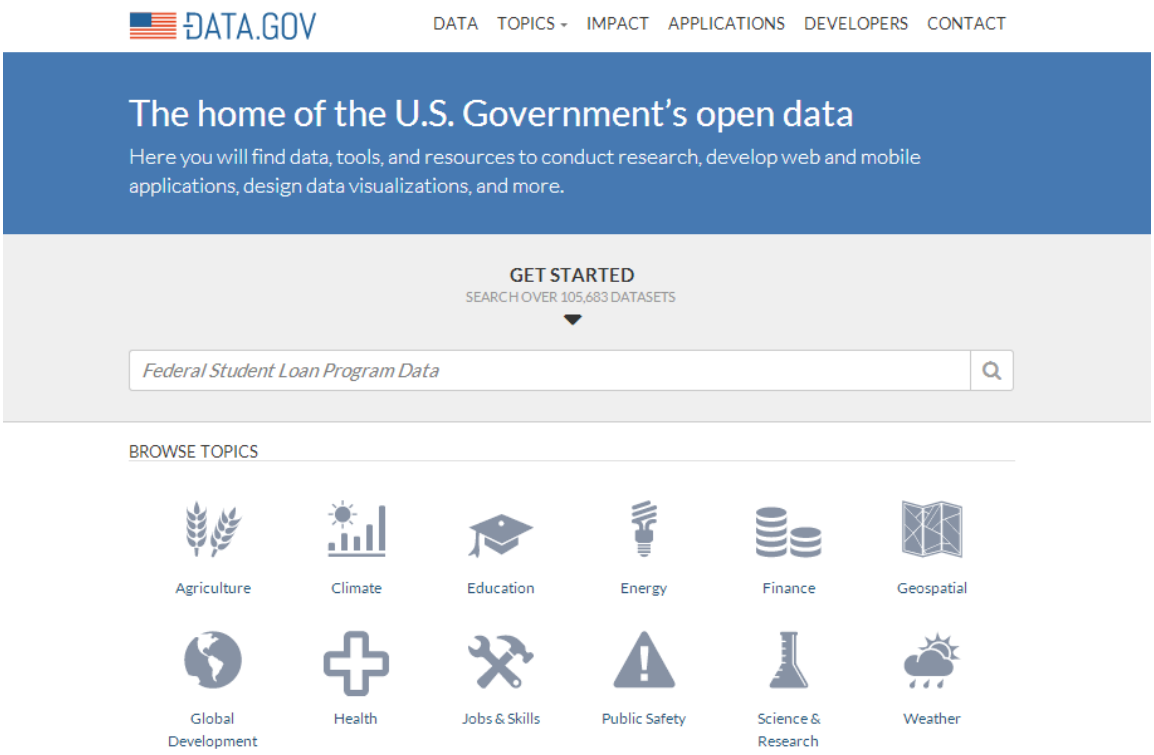


Figura 2.1 – Página principal del portal Open Data de los EE.UU.

Como es de esperar, en él se encuentran conjuntos de datos de todo tipo con información que gestiona el Gobierno de los Estados Unidos como la agricultura, el clima, los negocios o las ciudades, entre otros. Lo único que tiene que hacer el usuario es elegir una categoría y explorar los conjuntos de datos por sí mismo, o hacer una búsqueda más concreta a través de un formulario que incluso nos sugiere palabras clave o consultas comunes.

Data.gov está desarrollado con **CKAN**, un proyecto de la *Open Knowledge Foundation* y que sin duda es un poderosísimo *framework* para plataformas de datos de código abierto, muy popular internacionalmente y con un equipo de desarrolladores pioneros en sistemas Open Data. Incluye una robusta API que ha sido extendida por los desarrolladores de este proyecto para sacar el máximo partido a los conjuntos de datos. De esta manera resulta muy práctico utilizar la información que se ofrece y desarrollar todo tipo de aplicaciones por parte tanto de empresas como de ciudadanos.

Existen actualmente un gran número de "apps" desarrolladas por terceros (entidades, ciudadanos, etc.) que usan datos procedentes de data.gov y que podemos encontrar en su sección de aplicaciones.

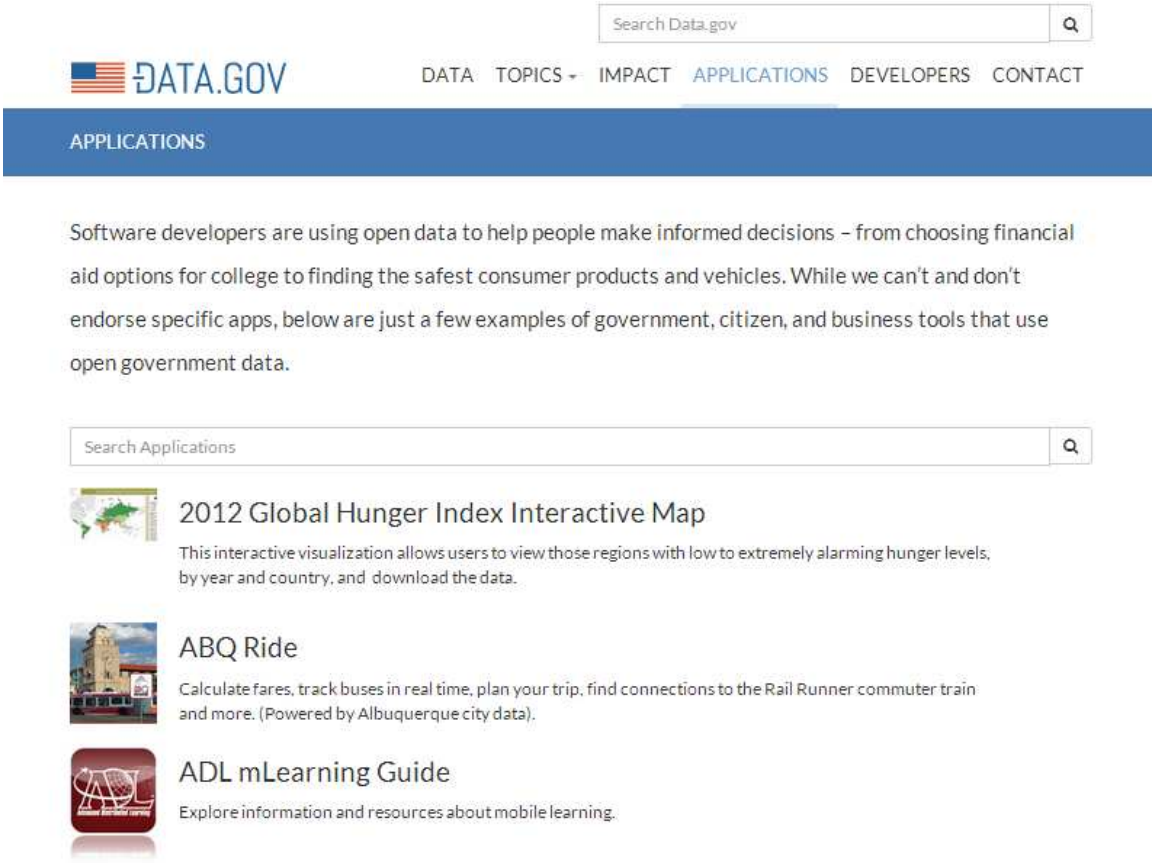


Figura 2.2 – Aplicaciones que usan el Open Data de los EE.UU.

Queda destacar que el código de esta plataforma Open Data es totalmente libre y se encuentra disponible en repositorios alojados en la plataforma GitHub con licencia GNU. Esto es porque el Gobierno de Estados Unidos, siguiendo la filosofía de datos y gobierno abierto, promueve la participación por parte del ciudadano en el desarrollo y la mejora de su sistema.

❋ Open-data.europa.eu

Se trata del portal de datos abiertos de la Unión Europea [11], que consiste en un punto de acceso único a una extensa y variada selección de datos elaborados por instituciones y otros

organismos de la UE. Fue publicada una primera versión el pasado 1 de diciembre de 2012 por la Oficina de Publicaciones de la Unión Europea, y se encuentra disponible en las 24 lenguas oficiales que existen en la UE.

El sitio remarca que, generalmente, los datos se pueden utilizar, reutilizar, enlazar y redistribuir de manera totalmente gratuita con fines comerciales o no comerciales, siempre que se indique la fuente. No obstante, hay un pequeño número de conjuntos de datos que tienen unas condiciones especiales de utilización, normalmente relacionadas con la protección de derechos de propiedad intelectual de terceras partes, y en cuyo caso podremos encontrar un enlace a dichas condiciones.

Figura 2.3 – Página principal del portal Open Data de la UE

Este Open Data presenta los conjuntos de datos en forma de datos vinculados (*Linked Data*), que es una manera normalizada de representar datos y facilitan la conexión de información de diferentes fuentes. La web ofrece un terminal para realizar consultas sobre sus conjuntos de datos a través de SPARQL, y también informa sobre un vocabulario de metadatos que han creado mediante los vocabularios del catálogo de datos (DCAT) y del *Dublin Core* (DCT), cuya finalidad es ser compatible con el *"Asset Description Metadata Schema"* (ADMS). De esta manera, han logrado un sistema de datos vinculados

normalizados cuya interoperabilidad facilita a los desarrolladores buscar, encontrar y descubrir datos desde un único punto de acceso.

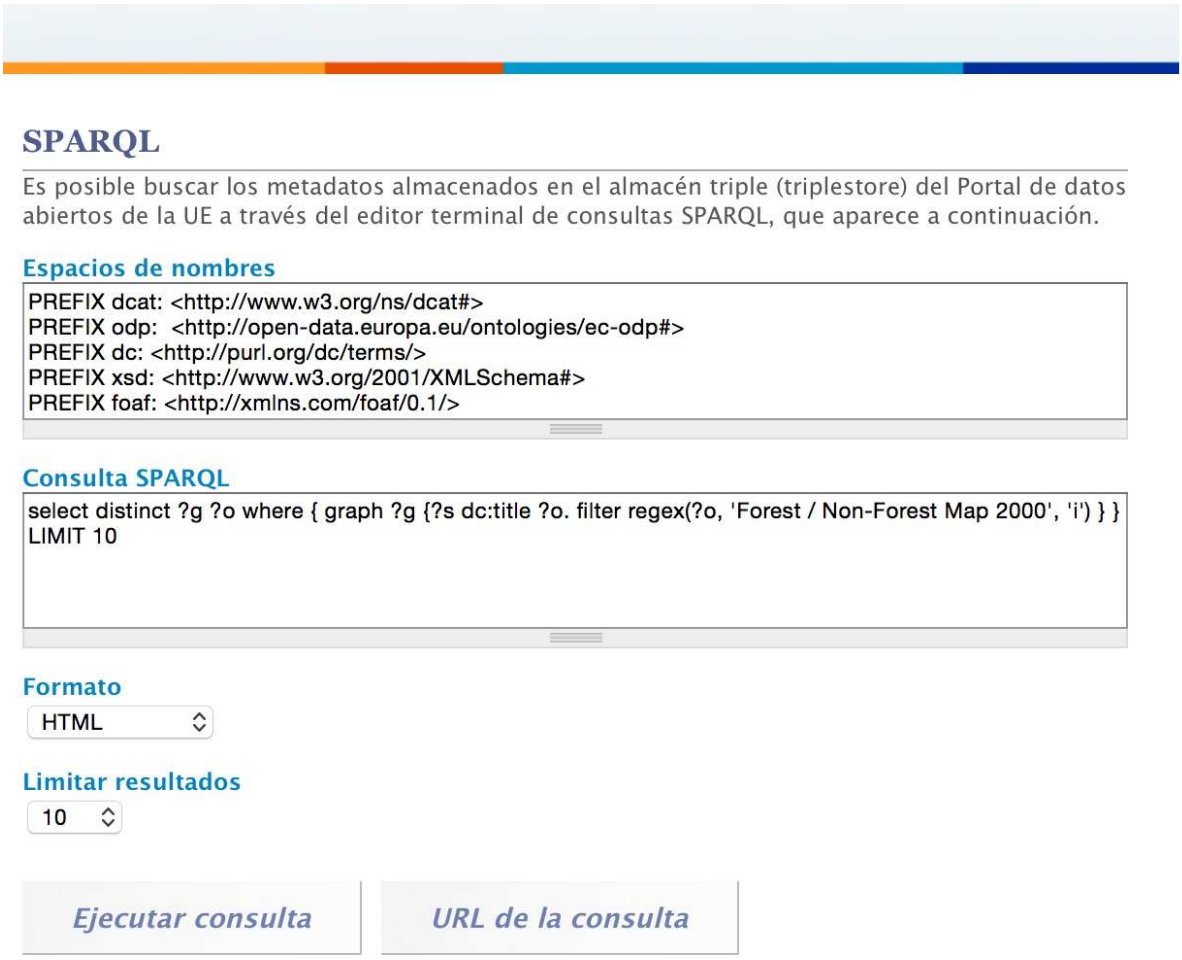


Figura 2.4 – Buscador de datos del Open Data de la UE

❋ Emitter.ca

En Canadá, el ámbito del medio ambiente se ha visto beneficiado por una iniciativa "open" con la creación de **Emitter** [12], un Open Data que proporciona a los ciudadanos canadienses los datos sobre la contaminación de las diferentes localidades del país.

En la siguiente figura podemos observar su página principal en la que nos explican qué información contiene y cómo usarla.

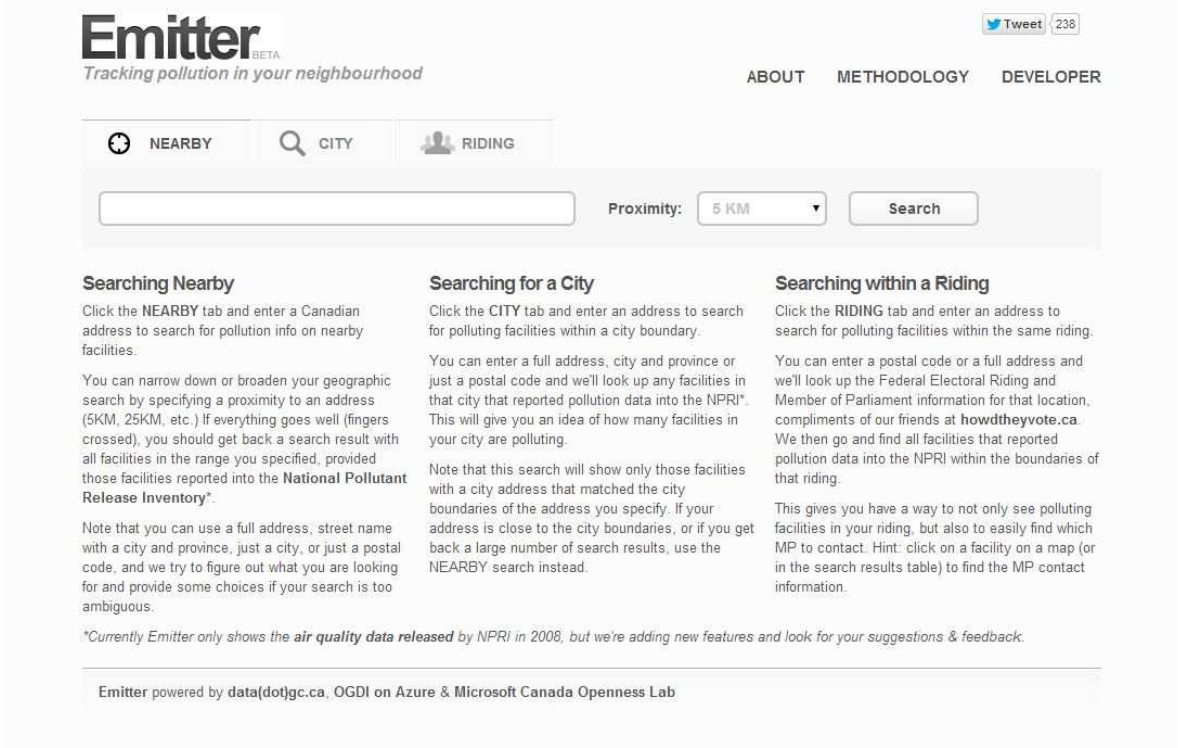


Figura 2.5 – Página principal del Open Data de Medio Ambiente de Canadá

El Open Data Emitter está desarrollado con CKAN, que ya conocemos de anteriores ejemplos, y **Drupal**, una plataforma de gestión de contenidos para la realización de sitios web y aplicaciones que además es "open-source" (de código libre).

Además, el equipo de Emitter ha desarrollado una API (que funciona a través del protocolo HTTP) para poder desarrollar aplicaciones con los datos que suministra. Su código es libre y está publicado en GitHub bajo una licencia BSD.

La figura 2.6 se corresponde con una búsqueda de los niveles de contaminación en la ciudad de Montreal. Como se puede observar, nos muestra un mapa con la localización de las diferentes estaciones en las que se pueden consultar los niveles de contaminación que se están recogiendo en ese momento.

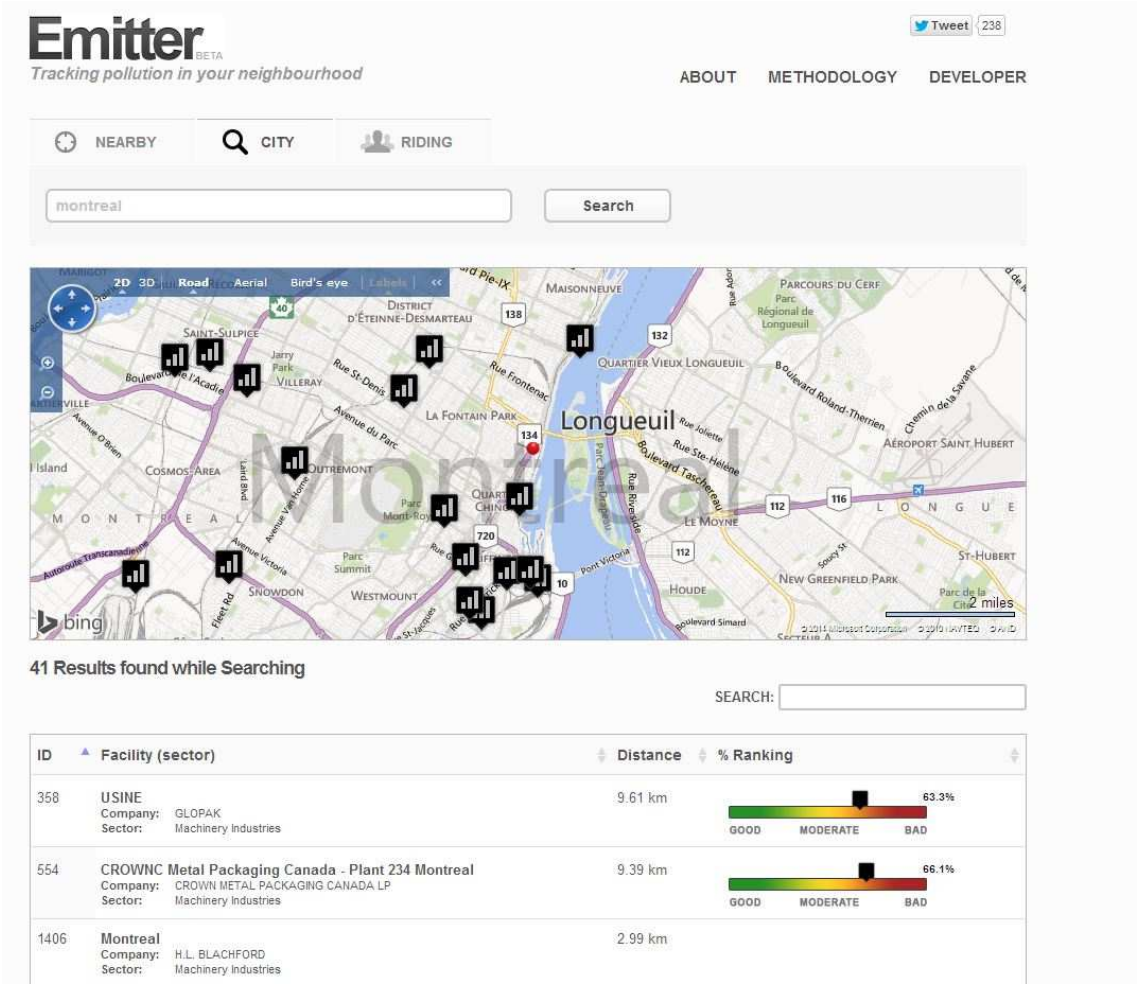


Figura 2.6 – Página principal del Open Data de Medio Ambiente de Canadá

❋ Datasf.org

El sitio datasf.org es el portal de datos abiertos [13] que se ocupa de dar servicio a la ciudad de San Francisco (EE. UU.).

Este Open Data es un buen ejemplo del sistema que, como explicaremos más adelante, queremos implementar, ya que aparte de mezclar las filosofías Open Data y Open Government, hace bastante hincapié en las soluciones que se pueden obtener haciendo uso de sus datos.

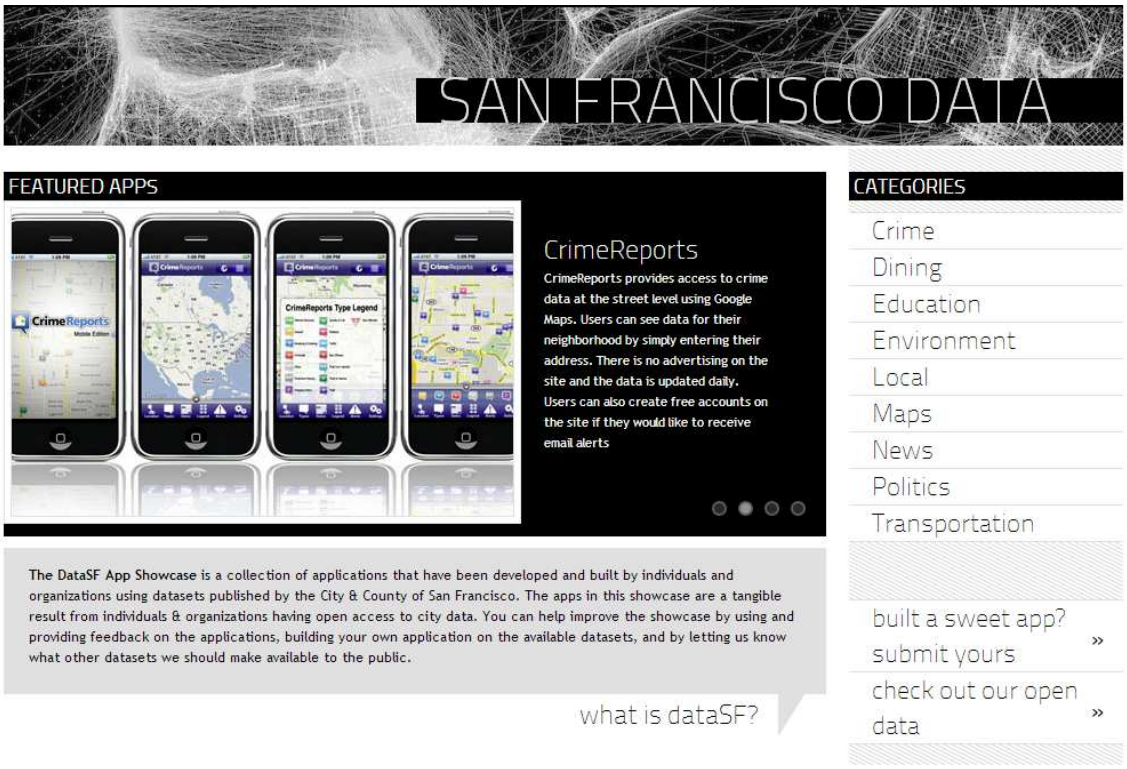


Figura 2.7 – Sección de apps desarrolladas a partir del Open Data de San Francisco

Además de publicar conjuntos de datos como los puntos de acceso Wi-Fi repartidos por la ciudad o, los emplazamientos de San Francisco más famosos por haber aparecido en películas o series de televisión, dispone de una sección en la que se exhibe el catálogo de aplicaciones que se han desarrollado basándose en datos del portal.

Algunas de las herramientas que utilizan el sitio web como fuente de datos son:

- **Crimespotting:** Muestra puntos en un mapa con las zonas de mayor inseguridad de la ciudad

- **Routesy San Francisco: A**plicación que informa en tiempo real donde se encuentra el medio de transporte más cercano para llegar a algún lugar deseado, asi como el tiempo que nos tomará dicho viaje.

- **EveryBlock.** Se trata de un filtro de noticias locales. Al proporcionarle el código postal de un distrito o barrio de la ciudad da información acerca de las llamadas realizadas a la policía, inspecciones de sanidad a los comercios, permisos de construcción, etc. Esta app en particular también funciona en las ciudades de Chicago, Washington o Nueva York entre otros.

- **The Original Parking Locator:** Sirve para localizar un coche aparcado. Indica datos relevantes como el lugar, la planta, el camino más corto para llegar a él.

❋ Otros destacados sistemas Open Data en el extranjero

- **data.gov.gh** – GODI (*Ghana Open Data Initiative*), Open Data del Gobierno de Ghana, publicado en febrero de 2012.
- **data.go.jp** - Open Data del Gobierno de Japón, publicado el 20 de diciembre de 2013, desarrollado con CKAN (al igual que los Open Data de los Gobiernos de Reino Unido o Australia).

2.2.2 Open Data en España

En esta sección vamos a mostrar alguno de los Open Data que podemos encontrar en la geografía española.

❋ Gobiernoabierto.navarra.es

El gobierno de Navarra lleva varios años desarrollando un proyecto que sigue la filosofía de gobierno abierto, el Portal de Gobierno Abierto de la Comunidad Foral de Navarra [14]. Han creado un portal en el que encontramos varias secciones, algunas de las cuales describimos a continuación:

Figura 2.8 – Sección "Participación" del Open Data del Gobierno de Navarra

- **Participación:** En esta sección se permite hacer propuestas al gobierno y se realizan encuestas informales y procesos de participación ciudadana formales en los que se podrán presentar alegaciones.

- **Open Data:** En esta sección se encuentra su sistema de datos abiertos, que básicamente se trata de un buscador y una serie de conjuntos de datos, presentados en diferentes formatos (como XML, XLS, CSV o RSS), proporcionados para su libre uso por parte del ciudadano y la creación de aplicaciones a partir de ellos. Además, podemos solicitar que se introduzcan nuevos conjuntos de datos que no están publicados.

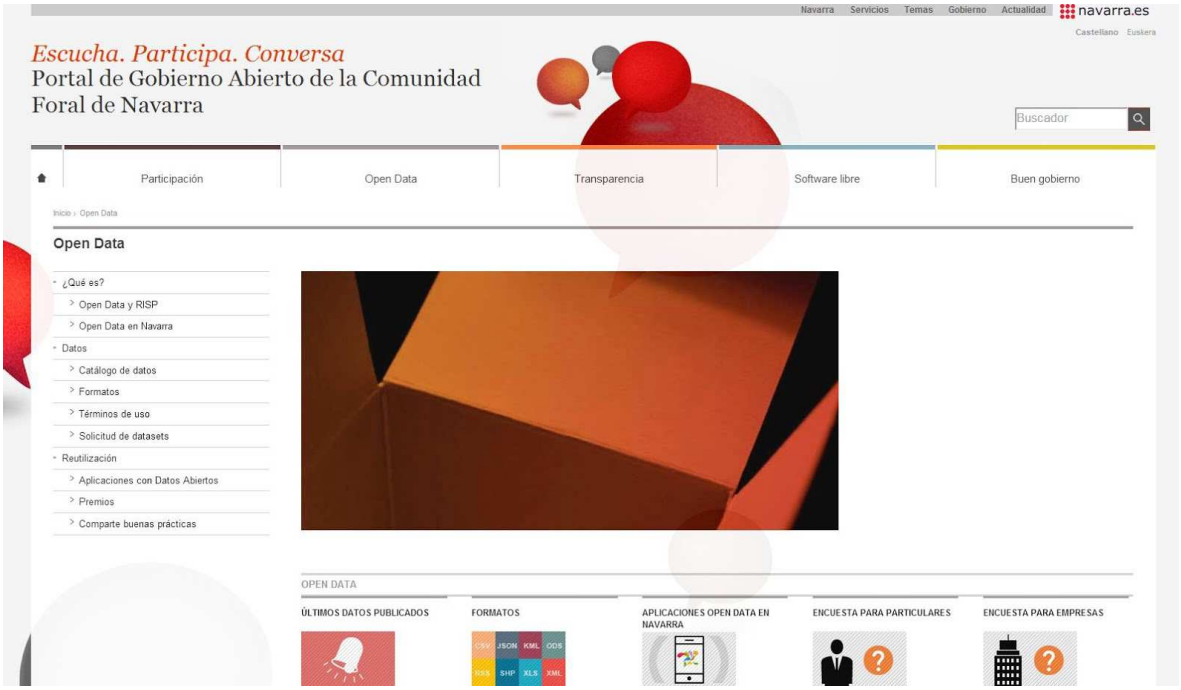


Figura 2.9 – Sección Open Data del portal del Gobierno de Navarra

- **Transparencia:** En esta sección encontramos las diferentes normativas que tiene el gobierno para ser más transparente. Además, encontramos varios documentos que contienen los presupuestos y los sueldos de los miembros del gobierno.

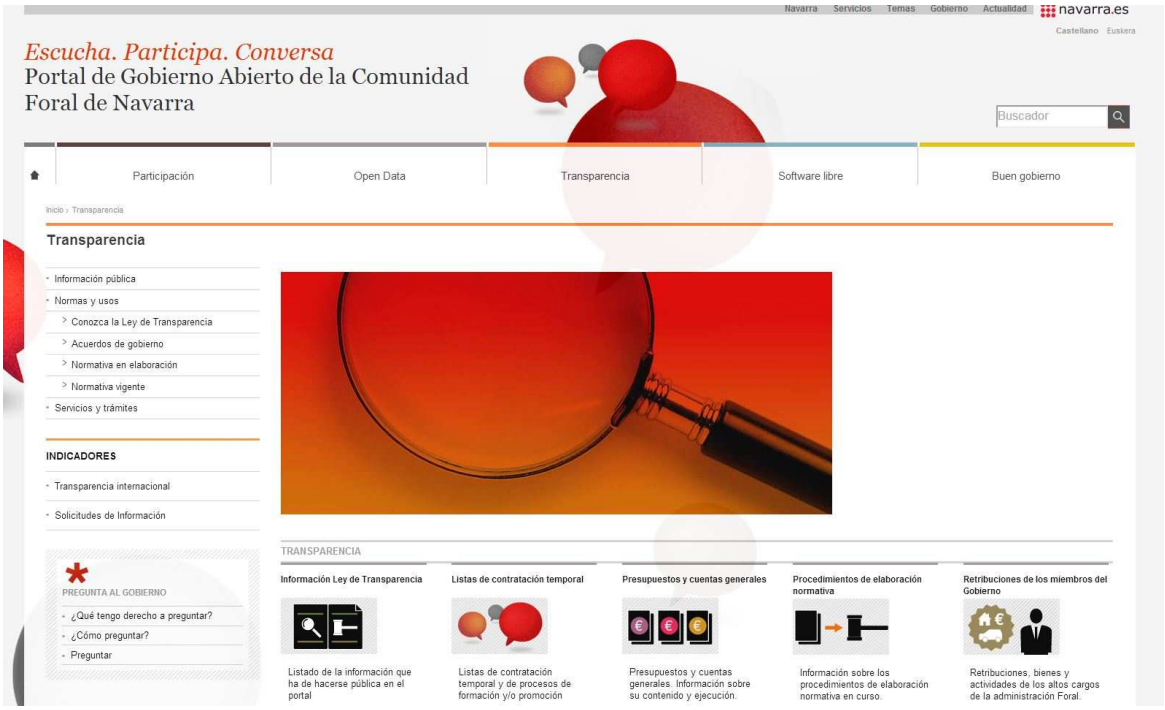


Figura 2.10 – Sección Transparencia del portal del Gobierno de Navarra

También poseen una sección dedicada al software libre, donde explican en qué consiste y en la que aseguran que el gobierno de Navarra utiliza herramientas de software libre. Sin embargo, su código no está publicado como código libre.

❋ Opendata.euskadi.net

Es el Open Data del Gobierno Vasco [15], y consiste en una web muy informativa donde expresan su compromiso a exponer los datos públicos que obran en su poder de forma reutilizable y bajo licencias de propiedad abiertas, dando una vez más pie a los ciudadanos a generar nuevos servicios. Contiene información muy completa sobre el sistema Open Data y sobre cómo hacer uso de los conjuntos de datos.

Tiene varias categorías de conjuntos de datos que dependen de su formato o forma y en consecuencia el conocimiento necesario para su uso. Para ciertas de estas categorías, Open Data Euskadi ofrece una API.

Además, ofrece al usuario ideas sobre la utilización de sus datos abiertos y ejemplos de diferentes aplicaciones que ya han sido desarrolladas a partir de ellos, así como una pequeña comunidad de intercambio de opinión donde se publican diferentes artículos.

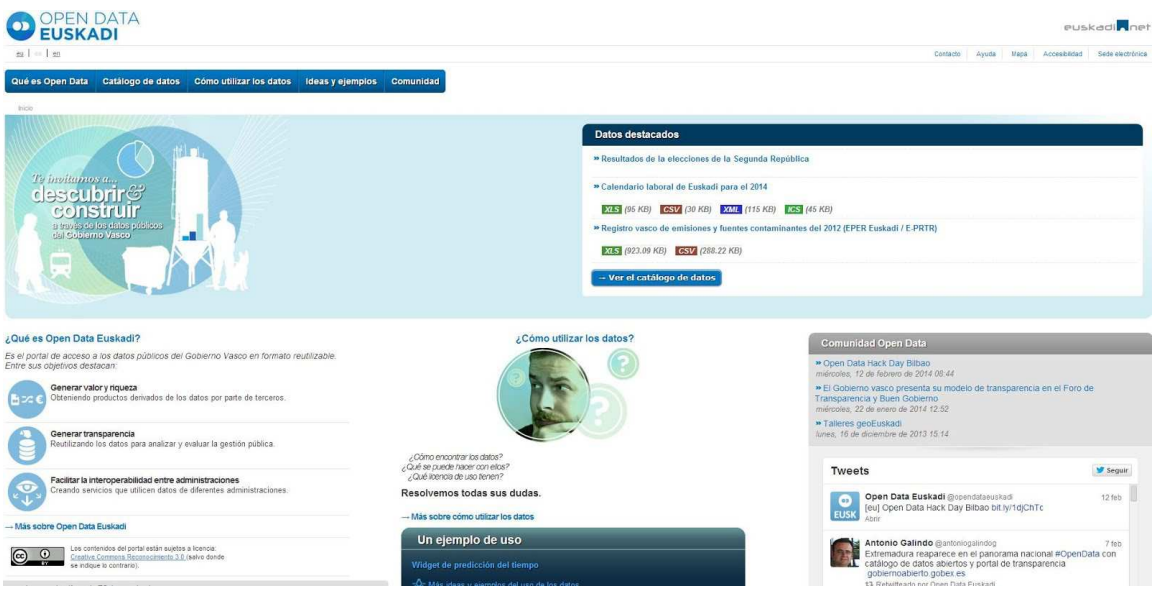


Figura 2.11 – Página principal del portal Open Data del Gobierno de Euskadi

Open Data Euskadi está muy relacionado con **Irekia (irekia.euskadi.net)**, la plataforma de desarrollo de Gobierno Abierto por excelencia en el País Vasco, a través de la cual el ciudadano puede participar en la vida política del país. Irekia sigue fielmente los tres pilares principales de la filosofía Open Government.

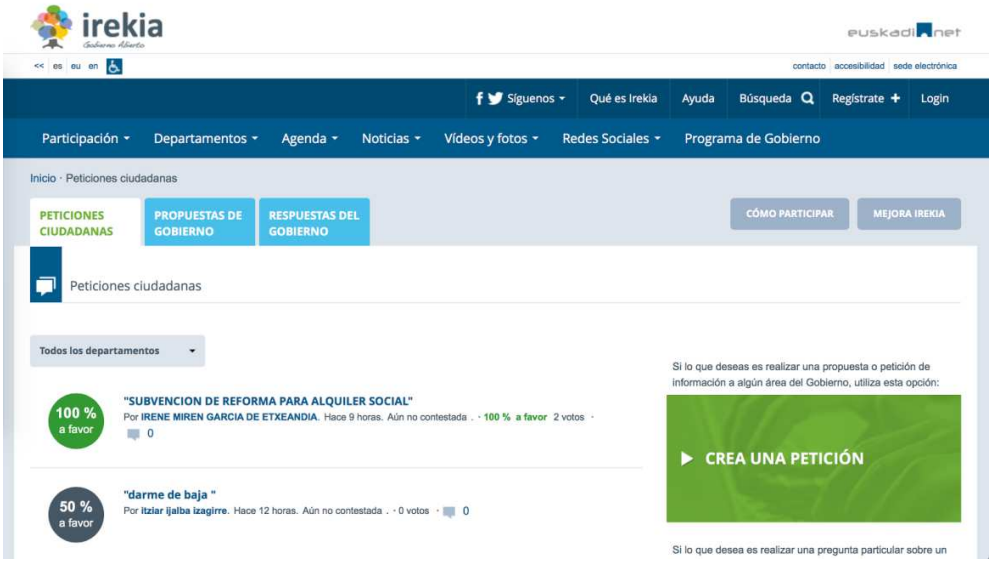


Figura 2.12 – Página principal de Irekia, la iniciativa de Gobierno Abierto de Euskadi

❋ Opendata.emtmadrid.es

En la ciudad de Madrid existe un Open Data de la Empresa Municipal de Transportes (EMT). Este sistema proporciona datos abiertos sobre la red de autobuses urbanos y otras líneas de la EMT [16]. Por un lado, ofrece conjuntos de datos "estáticos" sobre sus servicios en diversos formatos descargables, y por otro, nos ofrece datos abiertos "dinámicos" y en tiempo real. Para estos últimos, el Open Data de la EMT ofrece una completa API que facilita enormemente la integración por software para el desarrollo de aplicaciones; se tratan de datos de geolocalización de los autobuses en tiempo real, tiempos de espera, rutas en el mapa, entre otros.

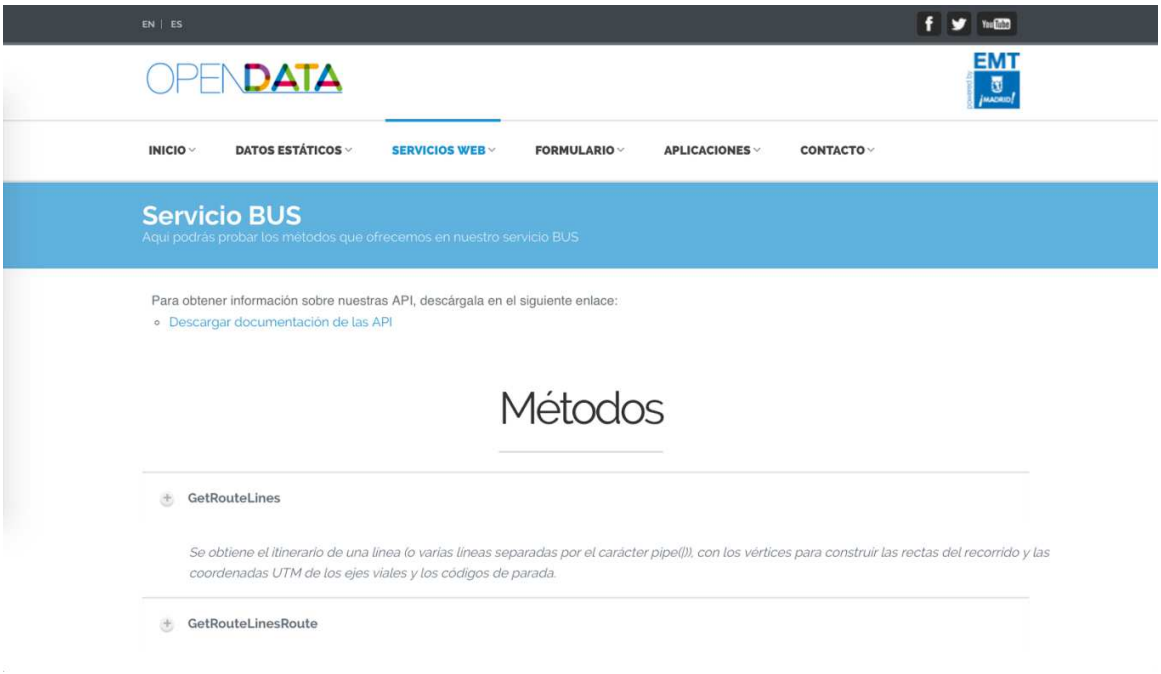


Figura 2.13 – API de llamadas disponibles en el Open Data de la EMT

Al facilitar el acceso a estos datos, este Open Data ha sido usado por muchos desarrolladores para poder crear aplicaciones web o aplicaciones móviles sobre los autobuses urbanos de Madrid, ofreciendo multitud de funcionalidades que serían imposibles de hacer sin los datos proporcionados por la EMT. Actualmente existe una "app" muy popular entre los habitantes de Madrid llamada "*Rapibús*" que informa sobre los tiempos de espera de una parada de autobús determinada. El

usuario tan sólo ha de introducir el número de identificación de la parada y la aplicación mostrará qué líneas de autobuses están por llegar y sus tiempos de espera.

❋ Datos.madrid.es

Datos Abiertos [17] es la reciente iniciativa del Ayuntamiento de Madrid, ligada como es de esperar a la filosofía de Gobierno Abierto, a través de la cual se ha creado un portal Open Data en el que informan sobre su actividad y desarrollo en este ámbito, y en el que se ofrece una amplia selección de conjuntos de datos que posee el gobierno madrileño y que deja a disposición del ciudadano.

El portal está desarrollado sobre Vignette (gestor de contenidos de carácter propietario). Los conjuntos de datos pueden encontrarse en diferentes tipos de formatos, desde algunos más simples y estáticos como XLS o TXT, hasta otros más complejos como RDF, los cuales proporcionan una serie de URIs para facilitar el acceso a los datos y realizar consultas sobre ellos.

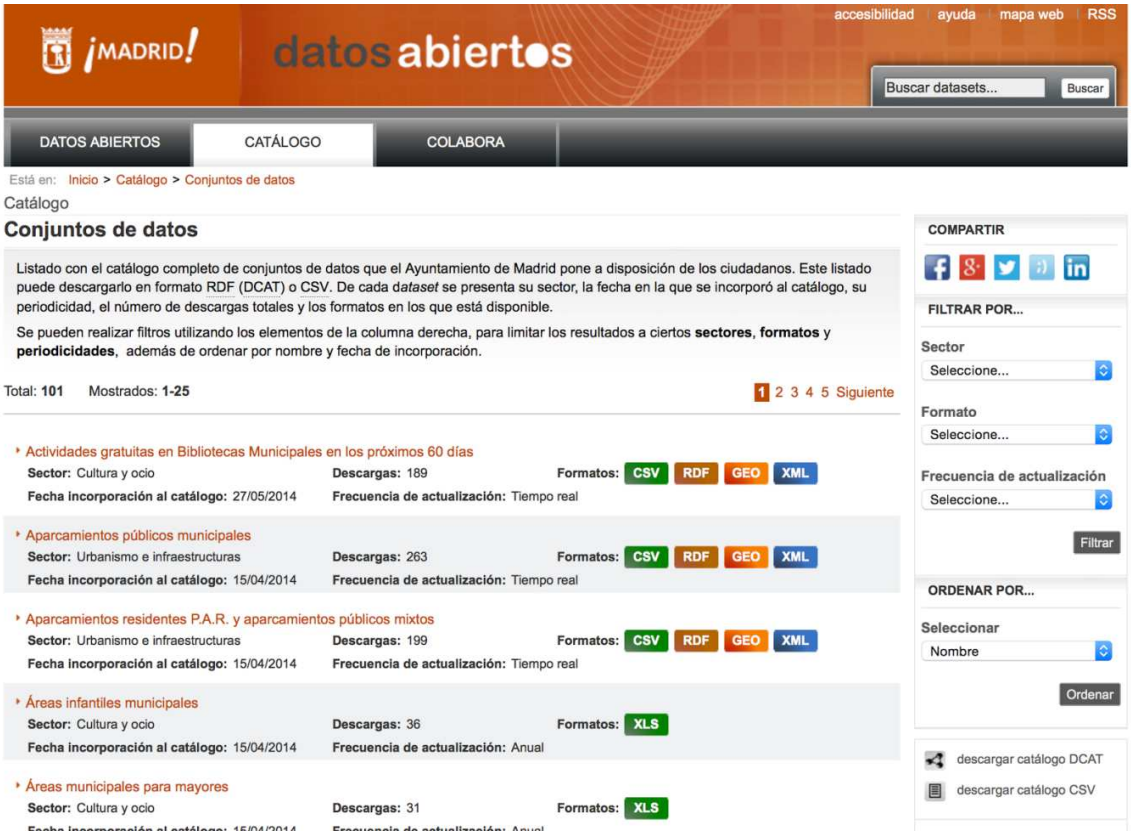


Figura 2.14 – Página principal del portal Open Data del Ayuntamiento de Madrid

Sin embargo, el Open Data de la Comunidad de Madrid no ofrece ninguna API, lo cual es poco atractivo para los desarrolladores ya que dificulta la integración software así como el desarrollo de aplicaciones complejas.

2.3 Las 5 estrellas de los Open Data

El 26 de Mayo de 2010, Tim Berners-Lee, creador de la Web, el lenguaje de programación HTML, el protocolo HTTP y el sistema URL, pionero en el *Linked Data* y *Linked Open Data* y, con el fin de motivar a la gente en el desarrollo adecuado de datos abiertos –especialmente a los propietarios de Government Data –, introdujo un sistema de valoración de 5 estrellas para plataformas Open Data [18]

En esencia, este sistema es una reutilización de la *pirámide de Maslow* [16]. En la primera estrella se recogen los puntos básicos y corresponde con la mínima calificación. Por el contrario, un portal Open Data con cinco estrellas (máxima calificación) significa que cumple todos los requisitos que se esperan de una herramienta de tales características. Con este sistema de calificación, se intenta promover que los organismos y entidades que usan tecnologías Open Data amplíen los horizontes de uso de dichas tecnologías.

Bajo el esquema de las estrellas, se consigue la primera estrella si la información está publicada, incluso si se trata de la fotografía de un documento de fax que se ha escaneado e impreso – siempre y cuando tenga una licencia de uso libre. De ahí en adelante se obtienen más estrellas a medida que mejoramos nuestro Open Data, haciéndolo más potente y práctico para el público.

Las características que definen el número de estrellas de un Open Data son:

- **Ninguna estrella** quiere decir que los datos no están disponibles bajo una licencia abierta, incluso si están disponibles on-line.

- **Una estrella** significa un buen comienzo: los datos son accesibles en la Web bajo una licencia abierta. Están en cualquier formato legible por el ojo humano, pero no por un programa informático. Esto suele deberse al uso de formatos de archivo propietario (imágenes, PDF…), por lo que no puede ser fácilmente reutilizado.

- **Dos estrellas** significan que los datos son accesibles desde la web en un formato estructurado legible por una máquina (software). Así, el consumidor puede procesar, exportar y publicar los datos fácilmente. Sin embargo, aún dependerá de software específico como Word o Excel.

- **Las tres estrellas** indican que los consumidores ya no necesitarán depender de software específico (como CSV en lugar de Excel, XML, entre otros). De acuerdo con esto, los datos podrán ser manipulados de cualquier forma, sin tener que ser destinados a un software particular.

- **Cuatro estrellas** quiere decir que ahora los datos no son accesibles a través de la web sino que "están en la Web", en formatos como RDF (*Resource Description Framework*) y abordables a través de URIs (*Uniform Resource Identifier*). Puesto que una URI es única, ofrece un control preciso y granular sobre los datos, lo que permite cosas como *bookmarking* (marcadores) y vinculación.

- Por último, las **cinco estrellas** indican que los datos no sólo "están en la Web" si no que además están vinculados a otros Open Data, explotando al máximo sus efectos en la red. A través de esta intervinculación, los datos son interconectados de una manera que crece exponencialmente, ya que se vuelven visibles a partir de otras fuentes y se da un contexto (por ejemplo, a través de links a Wikipedia).

Tim Berners-Lee afirma que cualquier conjuntos de datos, sea del formato que sea, es soportado por CKAN (proyecto de la *Open Network Foundation* del que Tim forma parte), y además CKAN proporciona los fundamentos necesarios para poder conseguir un Open Data de cinco estrellas.

III. Sistemas Open Data

Hasta ahora hemos hecho una introducción para poder abordar el tema del este proyecto y un Estado del Arte, en el que observábamos diferentes tecnologías utilizadas actualmente para la realización de iniciativas Open Data.

En un principio, el objetivo principal era el de adaptar una serie de aplicaciones desarrolladas el año pasado que utilizarían los datos contenidos del portal de Datos Abiertos del Ayuntamiento de Madrid. Debido a retrasos en el lanzamiento de dicho portal y que finalmente éste no iba a ofrecer un *Web Service* con una API de la que las aplicaciones externas se pudieran aprovechar, se decidió reestructurar el proyecto. Definitivamente, necesitábamos crear nuestra propia solución Open Data para llevar a cabo el proceso de adaptación, lo cual derivó en reescribir una aplicación por completo en lugar de adaptarlas todas. Debemos destacar, que dicha reestructuración del proyecto provocó que invirtiéramos una importante cantidad de tiempo en documentarnos sobre una cantidad importante de software que, finalmente, nunca llegamos a usar, como por ejemplo:

- Cómo desarrollar **aplicaciones web con JEE.**
- **Servidores para aplicaciones JEE:** Tomcat, JBoss, IBM WebSphere Application Server (WAS), CloudBees.
- Como empaquetar aplicaciones JEE con **Maven.**

Así pues en este capítulo nos centraremos en desarrollar nuestro propio sistema Open Data y posteriormente, en el cuarto capítulo, abordaremos la segunda parte del proyecto que hace referencia a las Aplicaciones de Medio Ambiente vinculantes al proyecto de la UCM con el Ayuntamiento de Madrid.

Antes de comenzar con el diseño del Sistema Open Data y la propuesta de integración, queremos ofrecer una introducción que sitúe al lector en un contexto concreto que le ayude a entender la motivación principal de este proyecto y sus consecuencias.

3.1 Las apps del grupo G-Tec y el Convenio de Colaboración con el Ayuntamiento de Madrid

En la Facultad de Informática de la UCM existe un grupo de investigación denominado **G-Tec** (Grupo Tecnologías UCM) que está formado por docentes e investigadores. Los proyectos desarrollados por este grupo están orientados a las tecnologías móviles. Se centran

en el análisis del ciclo de vida de éstas, ofreciendo métricas de fiabilidad, rendimiento, etc., además de estudios de mercado y posicionamiento [19].

El grupo G-Tec posee un convenio con el **Ayuntamiento de Madrid**, en particular, con la sección de **Medio Ambiente**, por el cual la universidad desarrolla proyectos de investigación de sistemas informáticos que después puedan ser aprovechados por la casa consistorial. Como se puede imaginar, estos proyectos están centrados en la especialidad del grupo de investigación (tecnologías móviles) y enfocados en la temática del Medio Ambiente.

De esta manera, durante el curso 2012-2013, se desarrollaron en la Facultad de Informática seis aplicaciones móviles para la plataforma Android con el objetivo puesto en satisfacer los desafíos que desde la administración pública se planteaban. Las apps implementadas fueron las siguientes:

- **Itinerario por los Jardines del Buen Retiro**

  Se creó con la intención de poner al servicio de la ciudadanía un sistema de visita auto-guiada a través de los diversos puntos de interés de los jardines del Parque del Retiro. Incluye fotografías y locuciones de audio que hacen más entretenido el paseo aportando datos sobre el tipo de jardinería y relatando curiosidades históricas

- **Recycla.te**

  Este proyecto tiene como objetivo concienciar y ayudar a la ciudadanía, especialmente a los jóvenes, en los temas relacionados con el reciclaje. Concretamente, se centra en ayudar en el tema del separado de residuos. Para ello implementa un sistema por el cual se puede escanear un código de barras y la aplicación informa del contenedor donde se debe desechar dicho residuo. En el caso de objetos que no dispongan de código, el usuario rellena un formulario que servirá para clasificarlo según sus materiales. También posee un juego de tipo Trivial que hace más ameno el aprendizaje de los temas tratados en la herramienta.

- **Recycla.me**

  Esta aplicación es similar a la anterior, aunque esta se inscribe dentro de las aplicaciones didácticas orientadas a padres con hijos entre 8 y 12 años. La idea es que padres y niños, conjuntamente, descarguen la aplicación con el objetivo de sensibilizar a ambos y ayudar en el aprendizaje de la separación de los residuos domésticos. Posee también la característica de identificar los productos por código de barras, así como secciones de información, curiosidades, etc.

- **Hábitat Madrid**

  Con este proyecto se pretenden centralizar todas las actividades medioambientales como cursos, talleres, visitas guiadas, jornadas, etc., que organiza el Ayuntamiento de Madrid en una sola aplicación. De esta manera, los ciudadanos pueden informarse y además realizar reservas para dichos eventos a través de sus dispositivos móviles. Este proyecto se encuentra pendiente de finalizar su desarrollo.

- **Camino seguro al cole**

  Es una herramienta de apoyo a familias y colegios. Permite que los niños realicen el itinerario de casa a la escuela y viceversa de manera autónoma. La aplicación muestra cuales son las rutas más transitadas por los demás niños. En base a esto, padres e hijos pueden definir su propia ruta de manera que al final los niños hagan la ruta todos juntos de forma independiente. También permite a los padres realizar el seguimiento por ubicación de los hijos además de valorar e informarse de una serie de comercios denominados "amigos de la infancia". Estos locales se encuentran situados a lo largo de las rutas de los niños y colaboran con este proyecto ofreciendo su ayuda en caso necesario. Actualmente, se encuentra en testeo en unos pocos colegios designados por el Ayuntamiento.

- **Mapa de Recursos Ambientales**

  La sexta y última aplicación informa de los distintos recursos y servicios municipales disponibles en la ciudad de Madrid, mostrando en un mapa los más cercanos en base a la ubicación actual del usuario o una dirección proporcionada por el mismo. Los recursos se han agrupado en 6 categorías: parques y jardines, aparcamientos (coche, moto y bicicletas), puntos limpios (fijos, móviles y contenedores de ropa), estaciones de suministro de biocombustibles limpios (GLP, GNC, bioetanol y electrolineras), vías ciclistas (desglosadas en calles tranquilas y rutas ciclistas) y por ultimo Áreas de Prioridad Residencial (APR).

Todas estas aplicaciones necesitan alimentarse de datos. Estos se encuentran almacenados en seis bases de datos (cada app posee la suya propia) distribuidas a su vez en seis servidores independientes. Por lo tanto, todas siguen un modelo cliente-servidor, donde el cliente es el terminal móvil y el servidor es la máquina remota que se ha preparado para proporcionar los servicios web necesarios que gestionen las consultas a las bases de datos desde los dispositivos.

En cuanto a la procedencia de los datos, esta puede tener dos orígenes: por un lado tenemos aplicaciones que tan solo consumen datos, es decir, solo hacen lecturas a una base de datos nutrida con contenidos e información proporcionada por el ayuntamiento. Este es el caso de las aplicaciones *Itinerario por los Jardines del Retiro* o *Mapa de Recursos*

Ambientales entre otras. Por otra parte, tenemos aplicaciones como *Recycla.te* y *Recycla.me* o *Camino Seguro al Cole*, las cuales son ellas mismas las que generan los datos y van construyendo la base de datos para futuras consultas.

En cualquier caso hay que destacar que los datos contenidos en estos servidores están orientados a ser tratados únicamente desde las aplicaciones instaladas en los terminales móviles, dado que ninguna publica la API de llamadas del servicio web o dispone de un portal donde visualizarlos de manera cómoda y más legible al ojo humano.

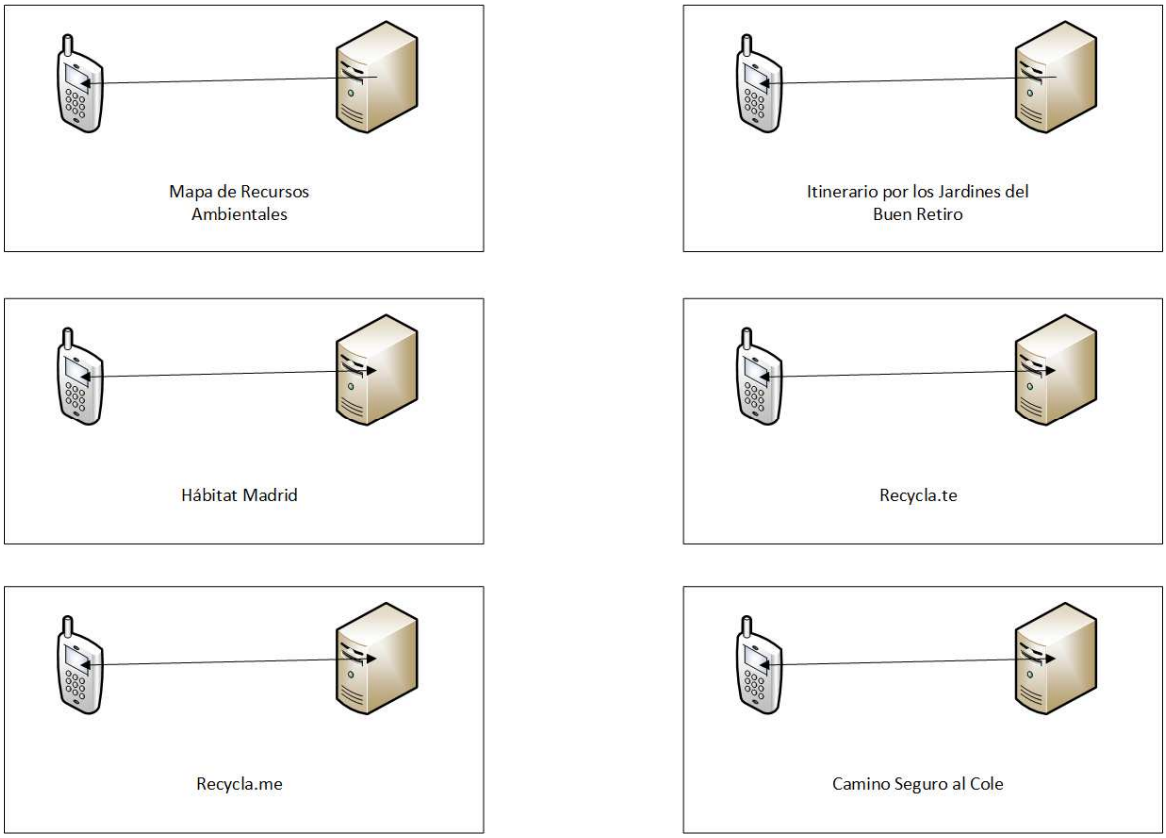


Figura 3.1 – Arquitectura aplicaciones G-Tec 2012-2013

Llegados a este punto, podemos proceder a fusionar las ideas que veníamos exponiendo desde el comienzo de esta memoria: los sistemas Open Data, junto a las iniciativas Open Government, con los problemas de consumo de datos de estas apps. Debemos aclarar que las aplicaciones en sí no tienen ningún problema y funcionan correctamente, pero parece natural pensar, que si las seis aplicaciones consumen o generan datos, que proceden o están destinados a ser usados por una administración pública como es el Ayuntamiento de Madrid, estos se encuentren unificados en un único sitio (un único servidor) que además

haga visible esta información sin restricciones, siguiendo las filosofías de datos y gobierno abiertos explicadas en el primer capítulo.

En el siguiente diagrama mostramos de una manera más gráfica la unión de los conceptos anteriores:

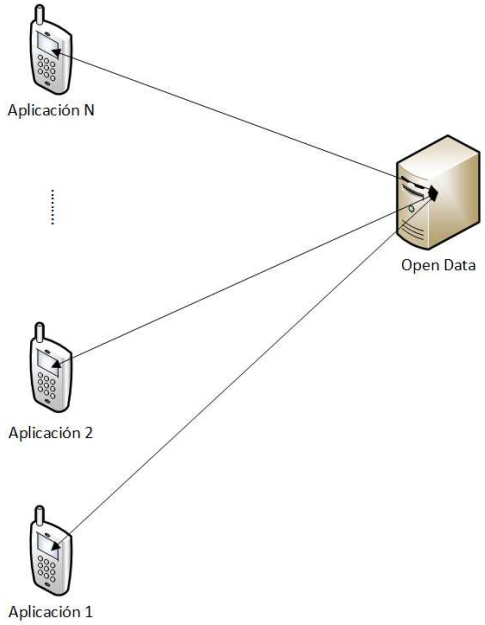


Figura 3.2 – Arquitectura aplicaciones G-Tec usando Open Data

Es así como surge el motor principal del presente proyecto. El objetivo final del mismo es diseñar, desarrollar y poner a disposición del público nuestro propio Sistema Open Data, el cual deberá ser capaz de satisfacer las demandas de las aplicaciones hasta el momento desarrolladas y, además, también servir para crear nuevas en el futuro.

En cuanto a la creación de nuevas aplicaciones no solo nos referimos a las apps dirigidas a los dispositivos móviles y orientadas a conexión, sino a todas las nuevas clases de herramientas que se puedan imaginar. El sistema a producir, además de disponer de algún servicio web para poder comunicarse con estas apps, también albergará otros accesos más sencillos como un portal de acceso simple que permita la visualización y la descarga manual de los conjuntos de datos en los que se basan dichas aplicaciones. Esto posibilita crear multitud de nuevas soluciones que procesan conjuntos de datos con el propósito de obtener algún resultado a partir de ellos.

Algunos ejemplos de estas nuevas iniciativas que se podrían crear son:

- Programas destinados a ser ejecutados en computadores convencionales. Estos programas pueden procesar los datos de dos maneras diferentes:
    - **Orientados a conexión:** A través de Internet mediante la API o servicio web.
    - **No orientado a conexión:** Descargando los ficheros en bruto para su posterior análisis de manera local.
- Páginas web que utilizan el servicio web para mostrar información en tiempo real basándose en datos del Open Data
- Sistemas RSS con actualizaciones de noticias, tráfico, etc.

Con todo esto, conseguimos ampliar el abanico de posibilidades y salidas que le podemos dar al Sistema Open Data, pero aún no hemos mencionado en qué conjuntos de datos se va a centrar dicho sistema. Este tema en concreto será tratado en profundidad en el capítulo siguiente, aunque por el momento podemos adelantar que la meta es desarrollar un Sistema Open Data enfocado al Open Government y cuyo objetivo es la publicación de conjuntos de datos propiedad de la administración pública (en concreto del Ayuntamiento de Madrid). Inicialmente, los datos iniciales pertenecerán al área de Medio Ambiente, aunque esto no es impedimento para añadir en un futuro otros conjuntos de datos pertenecientes a otras secciones (como los censos, el tráfico, los colegios, etc.) y completar el Open Data del Ayuntamiento.

Nuestro objetivo es crear una simulación y propuesta de integración de sistema Open Data para el Ayuntamiento de Madrid, que permita adaptar las apps desarrolladas en el grupo G-Tec y, utilicen dicho sistema en lugar de los servidores independientes que en este momento están usando.

Como punto de partida para orientar el proyecto, se ha elegido la aplicación Mapa de Recursos que nos ayudará a ilustrar los primeros resultados y ejemplos de uso. Por lo tanto, los conjuntos de datos iniciales del Sistema Open Data serán los necesarios para que dicha app funcione. El hecho de escoger esta entre las seis existentes obedece a la mayor adaptación de ésta a los estándares que establece el IAM —siglas de Informática del Ayuntamiento de Madrid; un órgano encargado, entre otras cosas, de validar el software antes de su liberación a la ciudadanía— en el desarrollo de aplicaciones móviles, aunque por el momento, esto es irrelevante para nosotros.

En resumen, durante las siguientes secciones vamos a realizar un análisis de las diferentes alternativas para levantar el Sistema Open Data junto a su portal y el servicio web asociado a él. Con ello pretendemos preparar una propuesta formal de integración que

recoja todas las características descritas en los párrafos anteriores. Después de presentar dicha propuesta, pasaremos a evaluar y resolver los problemas de alojamiento en servidores, tanto para nuestro Sistema Open Data como para la aplicación de Mapa de Recursos, pues antes de adaptarla a nuestro sistema, necesitamos ponerla en funcionamiento para analizarla y evaluarla.

Finalmente, en el capítulo siguiente nos concentraremos en describir detalladamente cómo integrar las aplicaciones con el Sistema Open Data, así como los conjuntos de datos y los procesos de normalización necesarios que permitirán llevar a cabo esta tarea.

3.2 Propuesta de integración

Nos documentamos sobre el concepto de Open Data y estudiamos las diferentes técnicas que existen para desarrollarlos. Las opciones que barajamos fueron desde elegir un gestor de contenidos para implementar el Open Data hasta desarrollar uno desde cero.

Finalmente decidimos usar un gestor de contenido de datos, denominado **CKAN.** Este gestor está especialmente diseñado para construir un Open Data, siendo uno de los mejores en su campo. Argumentamos las razones de dicha elección a continuación:

- Todo el código es abierto, incluyendo el núcleo de CKAN y todas las tecnologías necesarias para hacerlo funcionar, lo que encaja a la perfección con la filosofía de Open Data. Además, es gratuito y nos ofrece la posibilidad de contratar servicios especializados en caso de necesitarlos.

- Es una tecnología madura, ya que lleva años en el mercado y se utiliza en diversos Open Data que lo avalan.

- Nos garantiza que tendremos actualizaciones debido a que gobiernos de diferentes países contribuyen activamente a su desarrollo, por ejemplo Reino Unido y Estados Unidos.

- Podremos llegar a conseguir un Open Data 5 estrellas como lo ha hecho el Open Data de Cáceres, el cual usa CKAN.

- Existe la posibilidad de tener una API REST para cada conjunto de datos, con lo que favorecemos la creación de diferentes aplicaciones que reutilicen estos datos, como la aplicación que hemos diseñado de mapa de recursos para Android o Web.

3.2.1 CKAN

CKAN (siglas en inglés de *Comprehensive Knowledge Archive Network*) es un gestor de contenidos de datos que nos provee de las herramientas necesarias para gestionar el contenido de bases de datos y mostrarlas vía web [20].

Figura 3.3 – Logotipo de CKAN

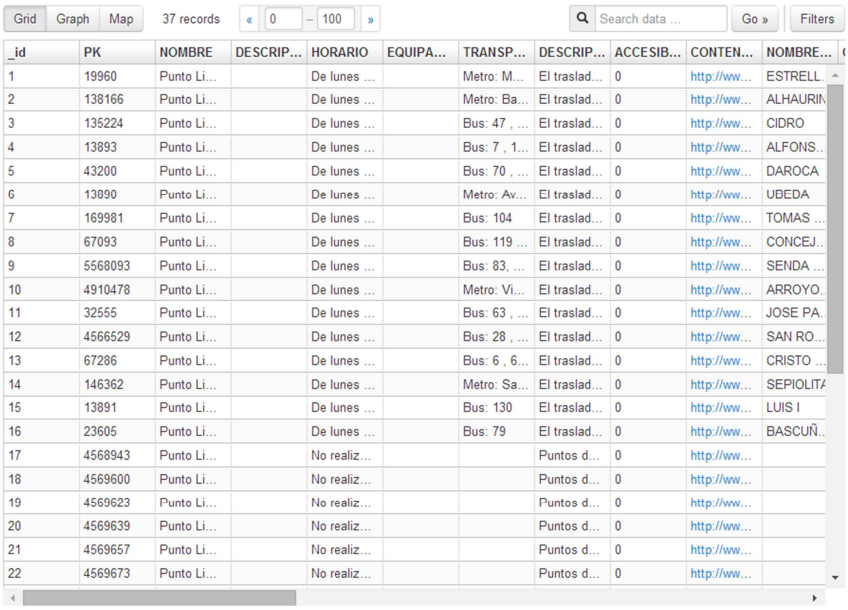
CKAN es de código abierto y tiene licencia GNU *Affero General Public License* (AGPL) v3.0, es decir, nos permite redistribuirlo y modificarlo siguiendo las directrices de dicha licencia, como por ejemplo, que deberemos nombrar a los creadores de librerías compatibles que se incluyan.

El gestor de contenidos CKAN nos proveerá las herramientas necesarias para crear un sitio Web con sus correspondientes bases de datos. A continuación enumeramos las diferentes funciones que tiene el Open Data que hemos desarrollado:

- Enlazar o subir directamente multitud de conjuntos de datos con diferentes formatos a nuestro Open Data.

Figura 3.4 – Formulario a rellenar cuando se sube un nuevo conjunto de datos

- Previsualizar el contenido de los conjuntos de datos directamente desde el sitio web sin tener que descargarlo.



Grid | Graph | Map | 37 records | « 0 – 100 » | Search data ... | Go » | Filters

_id	PK	NOMBRE	DESCRIP...	HORARIO	EQUIPA...	TRANSP...	DESCRIP...	ACCESIB...	CONTEN...	NOMBRE...
1	19960	Punto Li...		De lunes ...		Metro: M...	El traslad...	0	http://ww...	ESTRELL...
2	138166	Punto Li...		De lunes ...		Metro: Ba...	El traslad...	0	http://ww...	ALHAURIN
3	135224	Punto Li...		De lunes ...		Bus: 47 , ...	El traslad...	0	http://ww...	CIDRO
4	13893	Punto Li...		De lunes ...		Bus: 7 , 1...	El traslad...	0	http://ww...	ALFONS...
5	43200	Punto Li...		De lunes ...		Bus: 70 , ...	El traslad...	0	http://ww...	DAROCA
6	13890	Punto Li...		De lunes ...		Metro: Av...	El traslad...	0	http://ww...	UBEDA
7	169981	Punto Li...		De lunes ...		Bus: 104	El traslad...	0	http://ww...	TOMAS ...
8	67093	Punto Li...		De lunes ...		Bus: 119 ...	El traslad...	0	http://ww...	CONCEJ...
9	5568093	Punto Li...		De lunes ...		Bus: 83, ...	El traslad...	0	http://ww...	SENDA ...
10	4910478	Punto Li...		De lunes ...		Metro: Vi...	El traslad...	0	http://ww...	ARROYO...
11	32555	Punto Li...		De lunes ...		Bus: 63 , ...	El traslad...	0	http://ww...	JOSE PA...
12	4566529	Punto Li...		De lunes ...		Bus: 28 , ...	El traslad...	0	http://ww...	SAN RO...
13	67286	Punto Li...		De lunes ...		Bus: 6 , 6...	El traslad...	0	http://ww...	CRISTO ...
14	146362	Punto Li...		De lunes ...		Metro: Sa...	El traslad...	0	http://ww...	SEPIOLITA
15	13891	Punto Li...		De lunes ...		Bus: 130	El traslad...	0	http://ww...	LUIS I
16	23605	Punto Li...		De lunes ...		Bus: 79	El traslad...	0	http://ww...	BASCUÑ...
17	4568943	Punto Li...		No realiz...			Puntos d...	0	http://ww...	
18	4569600	Punto Li...		No realiz...			Puntos d...	0	http://ww...	
19	4569623	Punto Li...		No realiz...			Puntos d...	0	http://ww...	
20	4569639	Punto Li...		No realiz...			Puntos d...	0	http://ww...	
21	4569657	Punto Li...		No realiz...			Puntos d...	0	http://ww...	
22	4569673	Punto Li...		No realiz...			Puntos d...	0	http://ww...	

Figura 3.5 – Visualización de un conjunto de datos en forma de tabla en CKAN

- Si los conjuntos de datos que subimos tienen coordenadas, podremos situarlas en un mapa.

Figura 3.6 – Visualización de un conjunto de datos con coordenadas geográficas en CKAN

- Si los conjuntos de datos se cargan en formatos *CSV* o *XLS,* dispondremos de una *API REST* para realizar consultas a ellos

Figura 3.7 – API REST disponible para un conjunto de datos en CKAN

Para poder entender que es una API REST, necesitaremos comprender qué significan los conceptos API y REST, que explicaremos a continuación.

- Una API (*Application Programming Interface)* es un conjunto de llamadas o funciones que se ofrecen para establecer una comunicación con otro software, añadiendo así un nivel más de abstracción [21].

- REST (*Representational State Transfer*) es una técnica de arquitectura software para el desarrollo web que se apoya en el estándar *HTTP*. Nos permitirá crear un protocolo de comunicación sencillo con las operaciones *GET, PUT, POST, DELETE*, definidas por el estándar *HTTP* [22].

Una vez explicados estos dos conceptos, estamos en disposición de comprender que es finalmente una API REST. Esta no es más que un conjunto de llamadas o funciones para la comunicación con otro software (en nuestro caso CKAN). Dicha comunicación se realizará a través de Internet (mediante peticiones basadas en *URLs* junto con las operaciones antes descritas) desde una aplicación a un servidor, el cual responderá con un código de estado (*Status Code*) indicando si se pudo realizar dicha petición.

La API REST que utilizamos en nuestro Open Data está diseñada para devolver o modificar la información contenida en los conjuntos de datos. En las peticiones se puede usar un conjunto de funciones prediseñadas, como por ejemplo la búsqueda de un determinado elemento, o insertar código *SQL* para ejecutar las consultas en la base de datos. En cualquier caso, las respuestas devueltas siempre serán en formato *JSON*. En el capítulo 4 se explicará con más detenimiento un ejemplo de cómo realizar una consulta con la API REST al Open Data desde una aplicación móvil.

- Generación automática de archivos *RDF*, los cuales contienen las URIs e información pertinente de cada uno de los conjuntos de datos. Para generar automáticamente un archivo RDF a partir de un conjunto de datos, debemos escribir la siguiente *URL*:

http://<*dir. Open Data*>**/dataset/**<*nombre del conj. de datos*>.<*rdf* o *n3*>

Con lo que conseguiremos un archivo parecido al que se muestra en la siguiente imagen:

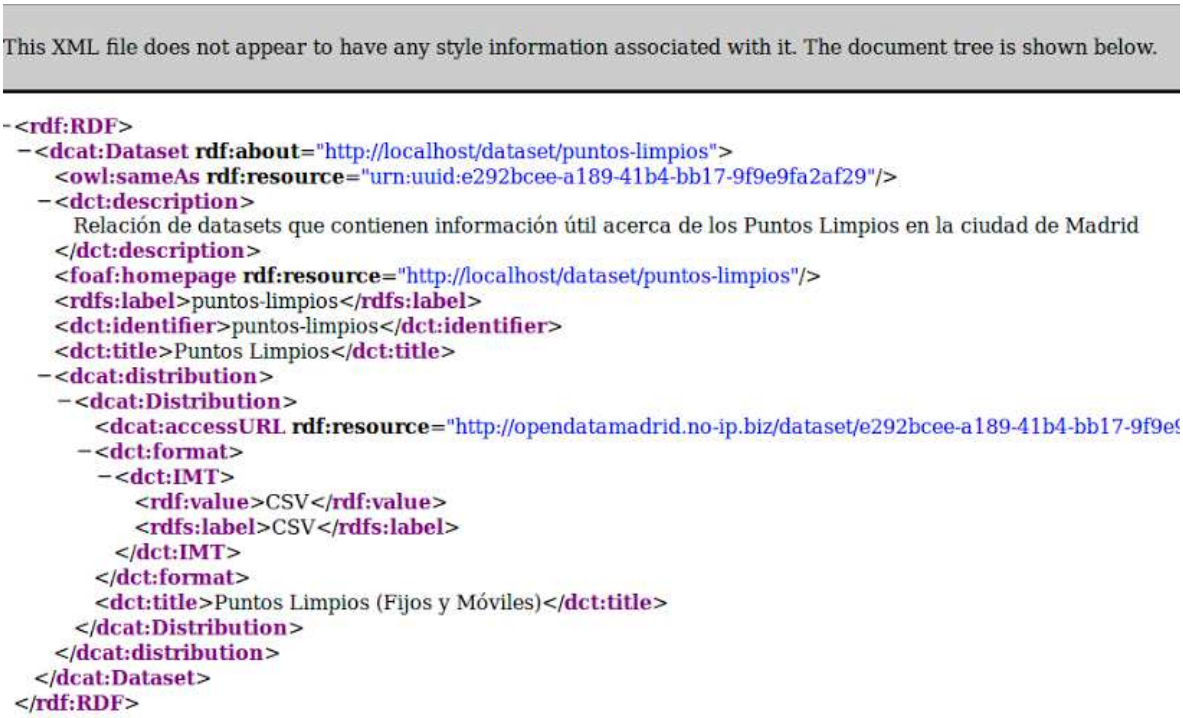


Figura 3.8 – Ejemplo de RDF generado automáticamente por CKAN

Gracias a esta última característica, la generación automática de archivos *RDF,* podremos alcanzar fácilmente una calificación para nuestro Open Data de 4 ó 5 estrellas.

Por último destacar que nuestro Open Data también tiene una serie de funciones que no son tan relevantes y que sólo enumeraremos a continuación:

- Registro para administradores, con sus correspondientes herramientas.
- Generación automática de estadísticas de los diferentes conjuntos de datos.
- Buscador de conjuntos de datos, organizaciones, etiquetas (*tags)*, etc.
- Comunicación con redes sociales: Facebook, Twitter, Google+.
- Traducción automática a multitud de idiomas disponibles.
- Creación de gráficas directamente en la página web a partir de los conjuntos de datos.

3.2.2 Tecnologías que usa CKAN

CKAN está desarrollado con diferentes tecnologías, que explicamos a continuación:

El *backend*, la lógica interna, está desarrollado en ***Python***. *Python* es un lenguaje de programación orientado a objetos, imperativo y que soporta programación funcional. Al mismo tiempo, es interpretado y usa tipos dinámicos. Se trata de un lenguaje multiplataforma. [23]

Figura 3.9 – Logotipo de Python

El *frontend*, la interfaz gráfica, está desarrollado en ***JavaScript***, ***HTML5*** y ***CSS***3 El *HTML* es un lenguaje de marcado para la creación de páginas Web, y es con el que se crea la estructura de la página Web. Los *CSS* son las hojas de estilo y con ellas daremos el formato a las páginas Web. Por último, el *JavaScript* es otro lenguaje de programación que se integrará con el *HTML* para convertir la página de estática a dinámica.

Figura 3.10 – Logotipos de CSS3, HTML5 *y* JavaScript

Para desarrollar CKAN se usa el ***Framework Web Pylons*** [24]. Este *framework* está escrito en *Python* y usa el estándar de *Web Server Gateway Interface (WSGI)*. El estándar *WSGI* [25] es un interfaz que nos indica cómo se debe comunicar el servidor web con las aplicaciones mediante el protocolo *HTTP*.

Para que el *framework Pylons* pueda implementar una aplicación que siga el estándar *WSGI* necesita un *middleware* que añade funcionalidad mediante sucesivas capas. Se suele explicar con un ejemplo cotidiano como es el de las capas de una cebolla, como se muestra en la figura 3.11.

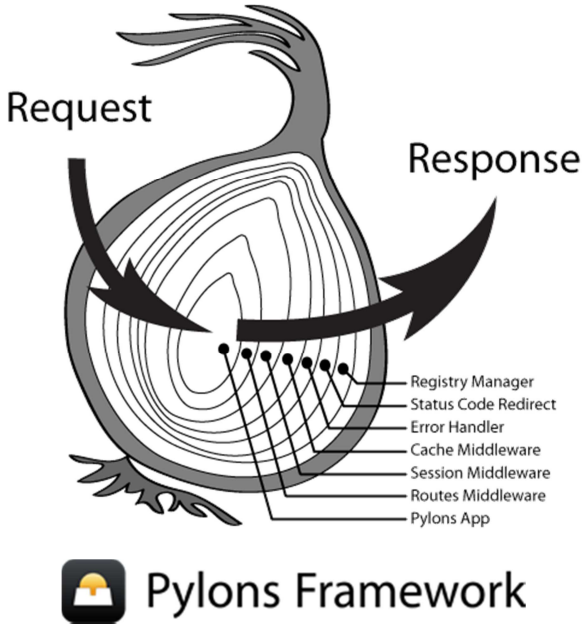


Figura 3.11 – Esquema por capas del framework Pylons

Además del *Framework Pylons*, se usan dos herramientas más: ***pip*** y ***virtualenv***. Sus objetivos son los siguientes:

- ***Pip*** sirve para que los diferentes paquetes software que se usan en un proyecto *Python* se instalen automáticamente y no se tengan que instalar manualmente.

- ***Virtualenv*** se utiliza para crear un entorno virtual de *Python* que pueda usar una especificación particular del lenguaje sin que esto afecte a cómo este configurado globalmente.

La arquitectura de CKAN utiliza el patrón **Modelo-Vista-Controlador** (o MVC). Esto es debido a que el *framework* está orientado para ser usado también con dicho patrón, entre otras razones, porque es el más común para el diseño de los sitios web. Este patrón sirve para estructurar la arquitectura software en tres módulos aislados:

- **Modelo:** se encuentran los diseños de las tablas que conforman las bases de datos.

- **Vista:** se encuentra la interfaz gráfica con la que interactuará el usuario.

- **Controlador:** se encuentra la lógica para darle funcionalidad a nuestra aplicación.

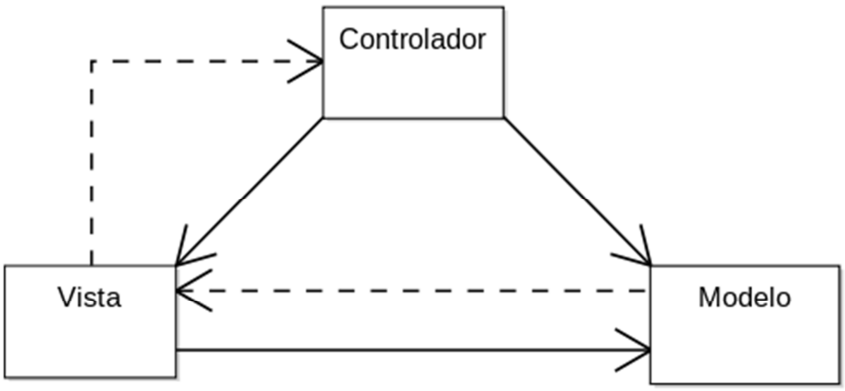


Figura 3.12 – Esquema del patrón Modelo-Vista-Controlador

Para el almacenamiento de los datos se usa una base de datos ***PostgreSQL.*** Se trata de un sistema de gestión de bases de datos relacionales que usa el estándar *SQL* para interactuar con ellas. Además, es también de código libre, siendo su licencia gratuita [26].

Figura 3.13 – Logotipo de PostgreSQL

Las transferencias de información entre la base de datos y el código en *Python*, utilizan la técnica de mapeo objeto-relacional (*ORM*). Esta se basa en mapear los tipos de un lenguaje orientado a objetos a unos tipos comprensibles por un gestor de base de datos relacional. En el caso del CKAN se usa ***SQLAlchemy*** [27].

Figura 3.14 – Logotipo de SQLAlchemy

Por último, para realizar búsquedas en las bases de datos, se usa ***Apache Solr***, el cual permite hacer *full-text search*; esto es, si proporcionamos un texto como parámetro en la búsqueda, devolverá como resultado todas las tablas de la base de datos que contenga dicho texto [28].

Figura 3.15 – Logotipo de Apache Solr

3.2.3 Configuración básica

En este apartado vamos a explicar cómo hemos configurado CKAN para convertirlo en nuestro Open Data. Resaltamos que esto no es una guía de instalación, sino más bien un documento donde explicaremos y argumentaremos por qué hemos elegido esta configuración.

Para crear nuestro Open Data se ha utilizado CKAN en su **versión 2.2**, que era la más moderna durante la realización de este proyecto. La principal razón de la elección de esta versión es que es de esperar que sea la más estable y con mayores funcionalidades.

La instalación se ha realizado sobre un ***Ubuntu Server 12.04* de 64 bits** ya que facilita bastante el proceso de instalación. Esto se debe a que el código fuente de CKAN se desarrolla para ser usado en este sistema operativo en concreto y por ello es posible encontrar copias ya compiladas del proyecto listas para usar. Sin embargo, es posible instalarlo en cualquier otro sistema operativo, aunque requiere como paso previo, el proceso de compilación particular para dicho sistema. El código fuente se puede obtener desde un repositorio público de GitHub [29].

A parte de instalar CKAN, hemos tenido que instalar software adicional necesario para el correcto funcionamiento de nuestro Open Data. Detallamos estas herramientas con el diagrama de la figura 3.16.

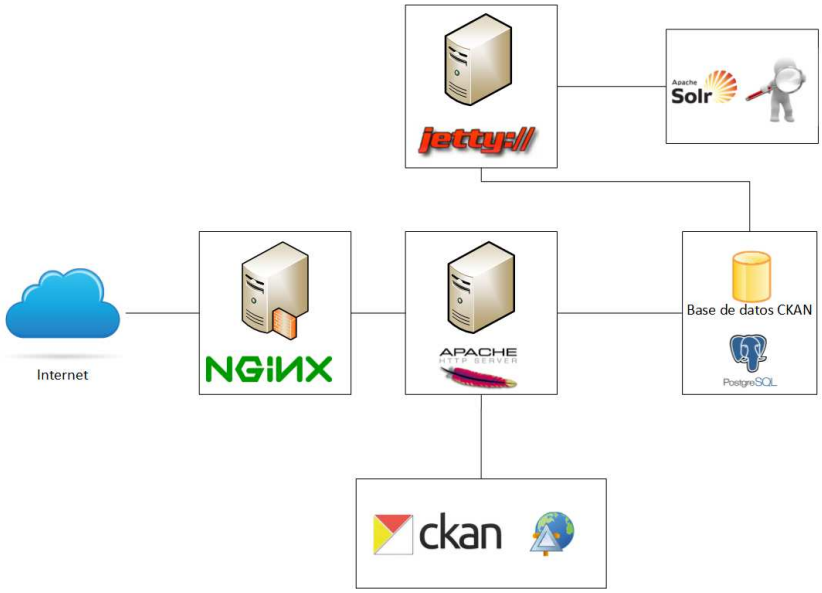


Figura 3.16 – Esquema detallado con todas las herramientas utilizadas en CKAN

En los siguientes apartados explicamos en qué consiste cada componente del diagrama.

❋ Nginx

Nginx es de código abierto y se puede usar con diferentes funcionamientos que son [30]:

- Servidor Web
- Proxy inverso
- Balanceador de carga
- Caché web

Figura 3.17 – Logotipo de NGINX

En nuestro Open Data lo hemos usado como proxy inverso, aunque al utilizar dicha funcionalidad, se incluyen implícitamente las de balanceo de carga y caché web. Para poder entender que es un proxy inverso, se deberá entender primero qué es un proxy, lo cual explicaremos a continuación.

Un proxy, también llamado servidor proxy, es un programa o un dispositivo que actúa de intermediario en una red. Por ejemplo, un cliente A quiere solicitar cierta información a un servidor C, pero en vez de pedírsela directamente a C se lo pide al servidor proxy B, y este será en realidad quien solicite la información a C. Con lo cual el servidor C nunca sabrá que el cliente A le solicitó alguna información.

Obtener la información de esta manera aporta ciertas ventajas como son, por ejemplo, añadir una capa de seguridad extra contra ataques de hackers, aumentar la velocidad de respuesta gracias a que tenemos una caché donde almacenar páginas temporalmente que evitan recargas innecesarias en el servidor. Además, cuando el proxy se encuentra conectado a varios servidores que hacen la misma función, podrá repartir equitativamente la carga entre ellos, lo que se conoce como balanceador de carga.

Así pues, un proxy inverso (*reverse proxy*) es un tipo específico de servidor proxy, que está situado entre uno o más servidores web y los clientes. El tráfico recibido desde Internet

por estos servidores es tratado por el proxy, el cual se encargará de hacer llegar las peticiones a los servidores evitando que los clientes se conecten directamente a estos [31].

❋ Servidor HTTP Apache

Apache es un software de código abierto que permite crear servidores web multiplataforma. Se trata del servidor web más usado en el mundo y también el que hemos utilizado para nuestro Open Data.

Figura 3.18 – Logotipo de Apache Server

En la figura 3.19 mostramos un diagrama de cómo sería la comunicación entre el servidor Apache y el cliente cuando actúa de por medio el proxy inverso Nginx:

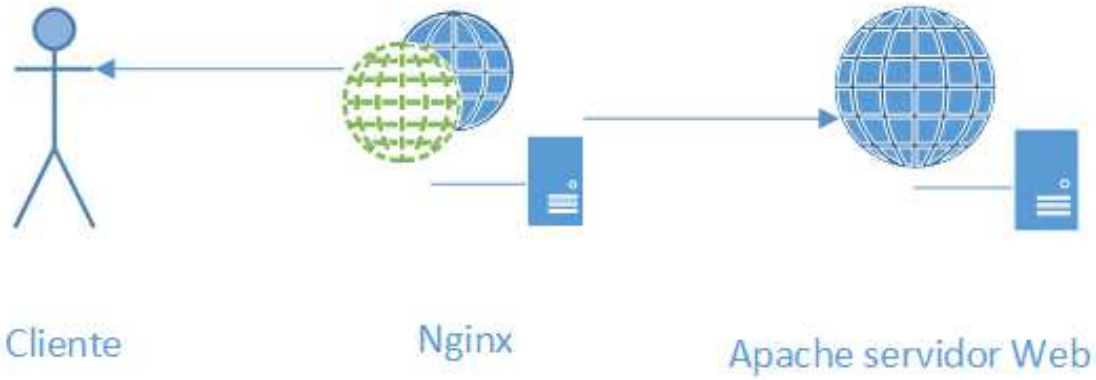


Figura 3.19 – Esquema de conexión Cliente-Apache Server con Nginx

❋ Las librerías modwsgi y libpq5

La librería ***modwsgi*** es necesaria para que el servidor Apache pueda comunicarse con el exterior y funcionar correctamente con aplicaciones *Python* que usan la interfaz *WSGI*. Como se ha explicado anteriormente CKAN está desarrollado con el *Framework Pylons* que utiliza el estándar *WSGI* [32].

Por último, la librería ***libpq5*** es una librería desarrollada en lenguaje *C* que sirve para poder comunicarse y hacer consultas a *PostgreSQL* desde una aplicación. En el caso de CKAN se utiliza para poder comunicarse con las bases de datos [33].

❋ Solr y Jetty

También hemos necesitado instalar ***Solr***, que como dijimos, sirve para poder hacer búsquedas en la base de datos, ya que nuestro Open Data tendrá un buscador. Para proceder con su instalación es necesario un servidor Web para aplicaciones desarrolladas en Java. Nosotros hemos decidido usar ***Jetty***, aunque se podría usar otro diferente. Nos decantamos por este mismo ya que se trata de software recomendado por los desarrolladores de CKAN debido a su sencillez y ser un proyecto de código abierto, siguiendo así la filosofía de los Open Data.

Una vez instalado *Jetty* y *Solr* debemos configurarlos. Para ello abrimos el archivo de configuración de Jetty, que se encuentra en */etc/default/jetty* y modificamos las siguiente líneas:

```
NO_START=0            # (línea 4)
JETTY_HOST=127.0.0.1  # (línea 16)
JETTY_PORT=8983       # (línea 19)
```

Para terminar, debemos sustituir el archivo *schema.xml* que viene por defecto en *Solr* por el que trae CKAN. Se puede hacer de la siguiente manera:

a) Creamos un *back-up* del archivo por si lo necesitamos revertir los cambios en un futuro. Esto se consigue mediante el siguiente comando:

```
# sudo mv /etc/solr/conf/schema.xml /etc/solr/conf/schema.xml.bak
```

b) Copiamos el archivo que está en el directorio del CKAN con el siguiente comando:

```
# sudo ln -s
/usr/lib/ckan/default/src/ckan/ckan/config/solr/schema.xml
/etc/solr/conf/schema.xml
```

❋ PostgreSQL

Por otra parte, hemos necesitado instalar *PostgreSQL* para el almacenamiento de los datos. Si quisiéramos podríamos usar otro gestor de bases de datos diferente, pero nos parece una muy buena opción usar éste, ya que se trata de los mejores gestores de bases de datos relacionales y además encaja perfectamente en la filosofía de Open Data al ser de código abierto, con lo que se tiene acceso a los archivos fuente y no se ha de pagar licencia alguna como se dijo anteriormente.

Una vez instalado, hemos creado una nueva base de datos para nuestro Open Data en la que se almacenarán los usuarios, las organizaciones, grupos, etc., pero en esta base de datos no se almacenará la API de los conjuntos de datos. Más adelante explicaremos el motivo de esta decisión.

Para la creación de la base de datos necesitamos un usuario. Podemos crear uno con el siguiente comando:

```
# sudo -u postgres createuser -S -D -R -P <Nombre de usuario>
```

Desde este momento, estaremos en disposición de crear la base de datos, aunque previamente hemos de puntualizar que las bases de datos que usa nuestro Open Data han de estar siempre en codificación UTF-8.

Comando para la creación de la base de datos:

```
# sudo -u postgres createdb -O <nombre del propietario> <nombre
base de datos> -E utf-8
```

❋ Archivo de configuración

Una vez hemos realizado los pasos anteriores, es necesario modificar el archivo de configuración que está en la ruta */etc/ckan/default* y que tendrá como nombre *production.ini* o *development.ini*, dependiendo de si nos encontramos ante un servidor de producción o de desarrollo respectivamente. En él tendremos que añadir la base de datos, el dominio, etc. Un ejemplo de configuración se puede encontrar en la figura 3.20.

Cuando hayamos terminado de realizar los ajustes en el fichero, nuestro Open Data podría empezar a funcionar de una forma básica, pero a nosotros no nos interesa que lo haga de esta forma, sino que queremos añadirle más funcionalidades que nos resultan indispensables. En el siguiente apartado denominado *plugins* explicamos cómo aumentar dicha funcionalidad.

production.ini

```
[app:main]
use = egg:ckan
full_stack = true
cache_dir = /tmp/%(ckan.site_id)s/
beaker.session.key = ckan

# This is the secret token that the beaker library uses to hash the cookie sent
# to the client. `paster make-config` generates a unique value for this each
# time it generates a config file.
beaker.session.secret = QiWUSYmtJJGVOdjo5qulOrSpe

# `paster make-config` generates a unique value for this each time it generates
# a config file.
app_instance_uuid = {55184504-c5e8-4c76-830c-bdd9728dd249}

# repoze.who config
who.config_file = %(here)s/who.ini
who.log_level = warning
who.log_file = %(cache_dir)s/who_log.ini


## Database Settings
sqlalchemy.url = postgresql://ckan_default:antonio25@localhost/ckan_default

ckan.datastore.write_url = postgresql://ckan_default:antonio25@localhost/datastore_default
ckan.datastore.read_url = postgresql://datastore_default:proyectoSSII@localhost/datastore_default
##para activar cual es le mas popular y demas

ckan.tracking_enabled = true

## Site Settings

ckan.site_url = http://opendatamadrid.no-ip.biz

#para activar app

ckan.dataset.show_apps_ideas = true


## Authorization Settings

ckan.auth.anon_create_dataset = false
ckan.auth.create_unowned_dataset =false
```

Figura 3.20 – Ejemplo fichero de configuración para servidor en producción

3.2.4 Plugins

Añadir más funcionalidades al CKAN se hace vía *plugins*, así es como lo denominan sus creadores. Estos *plugins* son extensiones que se añaden al CKAN, y cuyo desarrollo se puede llevar a cabo por el propio equipo de CKAN, por uno mismo o por terceras personas.

Los *plugins* están desarrollados en Python, siendo este el mismo lenguaje con el que se desarrolla CKAN. Además, cuando desarrollamos estos complementos, podemos hacer uso de la API interna que nos proporciona CKAN, la cual, entre otras cosas, permite acceder a la base de datos y consultar un grupo, un nombre de usuario, etc.

A continuación detallamos los *plugins* que vamos a usar, todos ellos han sido desarrollados por CKAN. Así nos garantizamos que tengan soporte en el futuro y si contiene errores serán arreglados.

❋ Plugin Stats

El plugin *Stats* viene activado por defecto en CKAN así que no tendremos que realizar ninguna instalación ni configuración.

Sirve para analizar la base de datos del Open Data y mostrar estadísticas, sobre el número de conjuntos de datos, número de organizaciones, etc. [34].

❋ Plugin text_preview

El plugin *text_preview* viene también activado por defecto en CKAN así que tampoco debemos realizar ningún paso adicional.

Sirve para poder previsualizar archivos subidos a nuestro Open Data, entre los que se encuentran XML, JSON, TXT, RDF, etc., es decir todos los archivos que sean de texto plano [35].

❋ Plugin recline_preview

El plugin *recline_preview* también viene activado por defecto en CKAN como los dos anteriores así que tampoco debemos hacer nada.

Sirve para poder tener una previsualización de archivos estructurados en nuestro portal de Open Data. Las extensiones que soporta son *XLS, CSV* o las tablas almacenadas por el *DataStore* en su base de datos.

❋ Plugin pdf_preview

El plugin *pdf_preview* no viene activado por defecto. Este plugin sirve para poder previsualizar, en el propio portal, los archivos PDF que se carguen en CKAN . Para activarlo tenemos que indicarlo en el archivo de configuración de CKAN. Escribiremos lo siguiente en dicho fichero:

```
ckan.plugins = <otros plugins>  pdf_preview
```

❋ Plugin DataStore

El plugin *DataStore* [36] no viene activado por defecto. Este plugin es uno de los más importantes en nuestro Open Data y de los más potentes que hay, ya que con él es el encargado de la generación de la API REST de los conjuntos de datos que se carguen al

portal, con lo que nosotros o terceras personas podrán utilizar la API para desarrollar nuevas aplicaciones que reutilicen estos datos.

El *DataStore* funciona gracias a una base de datos nueva que hemos tenido que crear para tal cometido en nuestro Open Data, esto significa que tenemos dos bases de datos, una para el *DataStore* y otra para el resto de la información de nuestro Open Data.

El *DataStore* provee una serie de funciones para poder crear nuevas tablas y añadir información a éstas. El único inconveniente de este plugin es que es el administrador el que tiene que añadir los datos por consola uno a uno. Pero esto tiene solución gracias a otro plugin llamado *DataPusher* que explicaremos en la siguiente sección.

A continuación explicamos cómo activar el *DataStore*:

- **Creamos un nuevo usuario en PostgreSQL para el *DataStore***. Este usuario tendrá solo permisos de lectura. Esto se consigue con el siguiente comando:

```
# sudo -u postgres createuser -S -D -R -P -l <nombre de usuario>
```

- **Creamos la base de datos para el *DataStore*.** El usuario propietario de la base de datos del *DataStore* debe ser el mismo que el de la base de datos de CKAN. Se realiza con el siguiente comando:

```
# sudo -u postgres createdb -O <usuario> ><nombre base de datos> -E
utf-8
```

- Añadir las siguientes líneas al archivo de configuración de CKAN:

```
ckan.plugins = <otros plugins> datastore

ckan.datastore.enabled = True

ckan.datastore.write_url = postgresql://<usuario propietario base
de datos>:<contraseña>@localhost/>nombre base de datos DataStore>

ckan.datastore.read_url = postgresql://<usuario lector creado
antes>:<contraseña>@localhost/<nombre base de datos>
```

- Por último, tenemos que poner correctamente los permisos. El siguiente comando permite ajustarlos:

```
# sudo ckan datastore set-permissions postgres
```

❋ Plugin DataPusher

El plugin *DataPusher* tampoco viene activado por defecto y no tiene sentido activarlo si no se utiliza junto con el plugin *DataStore*.

Como hemos indicado anteriormente, este plugin está íntimamente relacionado con el *DataStore*, debido a que usa las funciones de este último para la creación de tablas y llenado de las mismas.

Esta extensión lo que hace es, dado un fichero o una *URL* que contenga un archivo *CSV* o *XLS,* lo procesa. A continuación, utilizando las funciones que proporciona el *DataStore*, añade automáticamente toda la información a la base de datos. Este servicio es asíncrono porque el analizador sintáctico del fichero puede tardar un tiempo considerable. Otro detalle a destacar es que si fuera síncrono, podría bloquear el Open Data en una espera activa [37].

Para activarlo, debemos añadir las siguientes lineas al archivo de configuración de CKAN:

```
ckan.plugins = <resto de plugins> datapusher

ckan.datapusher.enabled = True

ckan.datapusher.url = http://0.0.0.0:8800/

ckan.datapusher.formats = csv xls

ckan.site_url = < Dominio>
```

❋ Plugin FileStore

El plugin *FileStore* es otro de los plugins que no viene activado por defecto. Esta extensión sirve para que podamos cargar cualquier tipo de archivo al gestor de contenidos. Esto se hace cuando se crea un nuevo conjunto de datos y puede realizarse de dos maneras: se enlaza el fichero con una URL al servidor remoto donde se encuentre o, subir una copia almacenada en nuestra máquina de manera local [38].

A continuación enumeramos los pasos necesarios para activar el *FileStore:*

- Primero tenemos que crear un directorio donde se almacenarán los archivos, eso se hace con el siguiente comando:

```
# sudo mkdir -p /var/lib/ckan/default
```

- Debemos añadir las siguientes líneas al archivo de configuración de CKAN:

```
ckan.storage_path = /var/lib/ckan/default

ckan.max_resource_size = < tamaño máximo de archivo>

ckan.max_image_size = < Tamaño máximo de imagen>
```

- Para finalizar, tenemos que establecer permisos al directorio anteriormente creado (conjunto de datos nuevo creado) en el servidor para que pueda guardar los diferentes ficheros. En nuestro caso usamos *Apache* sobre *Ubuntu*. Esto es importante saberlo ya que se necesita saber el nombre de usuario que tiene el servidor, en particular el nuestro es *www-data,* que suele ser el más común cuando se usa *Ubuntu* junto con *Apache*.

```
# sudo chown www-data /var/lib/ckan/default
# sudo chmod u+rwx /var/lib/ckan/default
```

3.3 El servidor del Open Data

En la siguiente sección vamos a explicar las herramientas software que utilizamos para montar el servidor que alojará nuestro Open Data.

Comenzaremos hablando del sistema operativo elegido para el servidor y a continuación explicaremos los dos servidores que hemos usado. El servidor de desarrollo que es donde se realizan las pruebas de configuración y el servidor de producción que es al que accederán los usuarios finales para usar nuestro Open Data en una versión pública y estable.

3.3.1 Ubuntu Server 12.04

Ubuntu Server 12.04 es un sistema operativo de la familia *Ubuntu*, basado en *Linux*, de código abierto, y desarrollado por la empresa Canonical, aunque la comunidad también puede participar en su desarrollo. Su licencia es totalmente gratuita [39].

Figura 3.20 – Logotipo del sistema operativo Ubuntu

Dentro de las diferentes distribuciones de Linux, nos decantamos por esta por encontrarse respaldada por una gran comunidad de desarrolladores y colaboradores. Aproximadamente, a día de hoy (2014), el 49% de todos los *Linux* instalados en máquinas son alguna versión de *Ubuntu*, por lo que se puede encontrar una extensa documentación sobre él.

La versión elegida es la 12.04 debido a que proporciona soporte extendido durante 5 años. Esta versión se liberó el 26 de abril de 2012 y su soporte terminará en abril de 2017, con lo que nos garantizamos actualizaciones de seguridad durante un largo periodo y se evitará tener que migrar a una nueva versión durante este tiempo. La nueva versión *LTS (Long Term Support)* de *Ubuntu* es la 14.04 y se liberó en abril de 2014, aunque no ha sido usada por ser lanzada después del comienzo de este proyecto.

Además escogemos la versión *Server* porque se instalará en un servidor desde el cual todo el mundo podrá acceder al sitio web del Open Data. De esta manera, optimizaremos recursos al no tener interfaz gráfica para el sistema operativo.

Por último escogemos la versión de 64 bits porque será más sencilla la implementación de nuestro Open Data (para más información consultar la sección anterior sobre CKAN).

3.3.2 Servidor de desarrollo CKAN

Un **servidor de desarrollo** es un entorno de pruebas, donde testearemos diferentes configuraciones en las que, por norma general, se producirán errores, razón por la cual no debe ser accesible a los clientes o usuarios finales. Además, la máquina de desarrollo no tiene por qué tener tantos recursos como una de producción, ya que no va a soportar una cargar regular de clientes. Tan sólo deberá tener los mismos recursos en caso de realización de pruebas de estrés.

La principal razón que ha hecho que estableciéramos un servidor de desarrollo ha sido la de tener un total control del sistema que probamos. Esto ha sido posible gracias a la disponibilidad de la máquina de uno de los integrantes del grupo para establecer dicho servidor.

A continuación detallamos cómo hemos construido el servidor de desarrollo usando ***Oracle VM VirtualBox***, el cual nos permitirá crear la máquina virtual que alojará el servidor.

❋ Oracle VM VirtualBox

Oracle VM VirtualBox es un software que nos provee de las herramientas necesarias para la virtualización.

La virtualización consiste en emular vía software un entorno concreto. Puede tratarse de la simulación de máquinas particulares, plataformas hardware o plataformas software como un determinado sistema operativo. En otras palabras, virtualizar es conseguir el comportamiento de algún entorno, hardware o software, sin disponer necesariamente del mismo ni nativamente ni físicamente [40].

Antes de distinguir los tipos de virtualización que existen, vamos a explicar que es una máquina virtual.

Una máquina virtual es un software que emula una determinada configuración de un computador. Esta simulación puede ser para imitar cierto hardware específico o una configuración de un determinado software [41].

Dentro de las máquinas virtuales nos encontramos dos tipos:

- Máquinas virtuales de sistema.
- Máquinas virtuales de proceso.

Nosotros vamos a usar las primeras. Se pueden tener varias máquinas de este tipo en un mismo computador y cada una de ellas tendrá su propio sistema operativo funcionando a la vez. A estas máquinas también se las suele llamar máquinas virtuales de hardware y se diferencian de las máquinas virtuales de proceso en que se ejecutan como un programa más en el sistema operativo. Suelen activarse automáticamente cuando algún tipo de proceso específico las necesita. Un ejemplo de este tipo es la máquina virtual de Java.

Podemos distinguir varios tipos de virtualización:

- Virtualización completa
- Virtualización parcial o *paravirtualización*
- Virtualización por Sistema Operativo

Para poder realizar el proceso de virtualización es necesario usar un software especializado, los denominados hipervisores (*hypervisor*) o monitores de máquina virtual (*virtual machine monitor VMM)*. El hipervisor es el encargado de monitorizar y gestionar dinámicamente los recursos de la máquina como la CPU, la memoria, la entrada/salida, etc. Esto es posible gracias a que se crea una capa que facilita la comunicación entre la máquina virtual y el hardware, haciéndole creer que tiene acceso universal a la máquina. El hipervisor, dependiendo del tipo que sea, se sitúa creando una capa entre el hardware y la máquina virtual que gestiona o, sobre un sistema operativo y la máquina virtual que gestiona [42][43].

Tenemos los siguientes tipos de hipervisores:

- **Tipo 1:** También llamado nativo o *bare-metal*. Se sitúa justo por encima del hardware, siendo el primero en cargarse antes que cualquier software o máquina virtual.

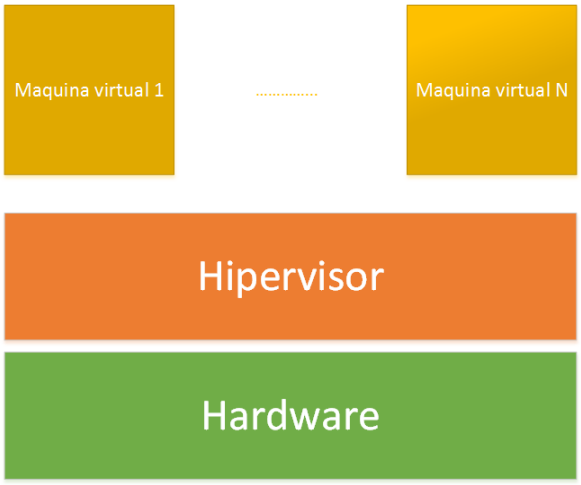


Figura 3.21 – Esquema hipervisor tipo 1

- **Tipo 2:** También se le llama alojado (*hosted*). En este caso, el hipervisor es un programa que se ejecuta sobre un sistema operativo como Windows, Linux, etc. El hipervisor solo interactuará con el sistema operativo.

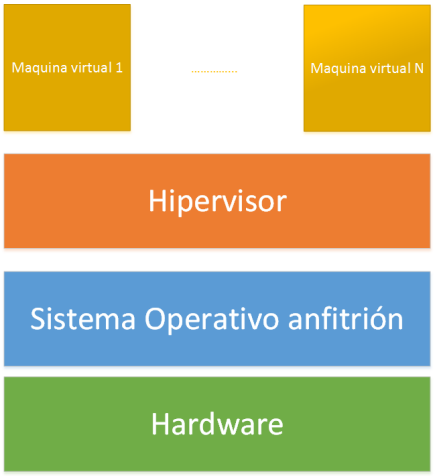


Figura 3.22 – Esquema hipervisor tipo 2

- **Híbridos:** El hipervisor interactúa con el sistema operativo y con el hardware a la vez. Esto es posible gracias a los procesadores X86, que incluyen en su arquitectura soporte para virtualización, lo que permite al hipervisor comunicarse directamente con el procesador. Estas extensiones se llaman IVT en los procesadores de Intel (*Intel Virtualization Technology:* Tecnología de Virtualización de Intel) y en AMD, AMD-V (*AMD Virtualization:* Virtualización de AMD) [44].

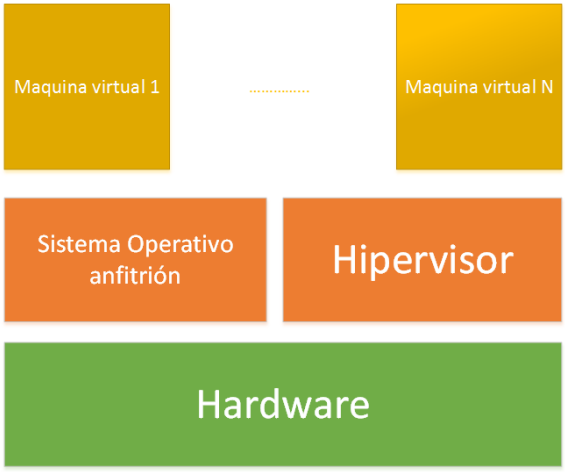


Figura 3.23 – Esquema hipervisor tipo híbrido

Nosotros vamos a usar como hipervisor *Oracle VM VirtualBox*, que es de tipo híbrido y cuyo tipo de virtualización es completa. Esta elección se debe a que queremos simular una máquina virtual que contendrá su propio sistema operativo y configuración específica [45].

Figura 3.24 – Pantalla de inicio de Oracle VM VirtualBox

Otra de las razones que nos empujó a utilizar este hipervisor es que es software gratuito con licencia GPL, y uno los ampliamente usados en la comunidad.

Esta herramienta nos facilitará las diferentes simulaciones para crear los entornos necesarios. Podemos crear multitud de máquinas, ya que una de las funcionalidades de VirtualBox es su capacidad para clonar máquinas en cualquier momento y conservar así su configuración. Esto supone un respaldo ante la realización de las diferentes pruebas de configuración, ya que, si una de ellas sale mal, simplemente se borra y se vuelve a una máquina clonada anterior a esa.

La versión de *Oracle VM VirtualBox* que vamos a utilizar es la 4.3.8, que era la versión más moderna cuando se empezó este proyecto. Esta se instalará en una máquina que funciona con *Windows 7 Professional de 64 Bits*, con lo que usaremos la versión que se adapta a este sistema operativo en concreto. No obstante, es posible obtener VirtualBox para para otras versiones de Windows y otros sistemas operativos, ya que ofrece soporte para diferentes versiones de *Linux* y *Mac Os*.

❋ Instalación de máquina virtual con Ubuntu Server 12.04

En esta sección detallaremos los parámetros de configuración más relevantes para la creación e instalación de una máquina virtual con *Ubuntu Server 12.04*, que usaremos como servidor de desarrollo.

Para empezar a crear la máquina virtual con *VirtualBox* tendremos que indicar que vamos a usar como sistema operativo un *Linux* y especificar que es un *Ubuntu de 64 bits*. Al ser de 64 bits podremos darle toda la memoria RAM que necesitemos, ya que teóricamente se podrían direccionar hasta 16 exabytes. Sin embargo, por cómo está diseñado *Ubuntu* de 64 Bits internamente, sólo soporta por el momento hasta un 1 TB de memoria RAM, lo que sigue siendo bastante más de lo que precisamos. Nosotros estableceremos 2GB de RAM, lo cual debería ser más que suficiente. En el caso de que fuera necesaria más RAM, sería posible añadirla cambiando configuración la configuración de la máquina y sin tener que reinstalar todo el sistema. Por último necesitaremos como almacenamiento no volátil (disco duro virtual) un mínimo de 15GB para poder instalar nuestros programas.

Antes de lanzar por primera vez la máquina virtual, necesitaremos realizar unos ajustes adicionales de configuración: en la sección de **Red** dentro del **Adaptador 1**, tenemos que seleccionarlo como **adaptador puente** y escoger la tarjeta de red que vamos a usar. Este paso es muy importante porque si no lo hacemos, por defecto se configura como *NAT*, y así no conseguiremos simular una tarjeta de red completa, la cual es necesaria para obtener una IP que haga accesible desde Internet el servidor que queremos montar. No obstante, de no realizar el paso, la máquina virtual si dispondría de acceso a Internet para navegar.

Una vez instalado el sistema operativo entraremos directamente en la consola ya que esta versión no incluye interfaz gráfica. Al ser una máquina de desarrollo le instalaremos un interfaz por comodidad, pero si fuera de producción, no sería recomendable hacerlo, ya que desperdiciaríamos recursos inútilmente. Nosotros instalaremos como interfaz gráfica *GNOME* aunque remarcamos que esto es completamente opcional, pudiéndose instalar cualquier otra o incluso ninguna.

A continuación hemos tenido que cambiar la configuración *IP privada* de la máquina virtual. Esto se hará modificando un archivo que se encuentra en la ruta: */etc/network/interfaces*.

IP (*Protocolo de Internet*) es un protocolo que permite la comunicación digital entre diferentes dispositivos. Una de sus principales funciones es el direccionamiento, el cual permite asignar direcciones a dispositivos y recursos en una red. Concretamente, una dirección *IP* es una secuencia de 4 bytes (*IPv4*), separadas por puntos que hacen referencia a la dirección de la interfaz de un dispositivo. Es equivalente a cuando se da la dirección de una casa para que se sepa a dónde tienen que dirigirse las cartas.

Además, las *IPs* pueden ser públicas o privadas. La diferencia que existe entre unas y otras es que la *IP pública* es la dirección que te da tu proveedor *ISP*, siendo esta la puerta de enlace por la que te conectas a Internet. Esta dirección es única. En cambio las privadas son las *IP* que se tienen dentro de una red local (LAN) o privada (VPN) para que se puedan distinguir unos equipos de otros dentro de dicha red. Existe un rango de direcciones prefijado para este fin, por lo tanto, estas direcciones no pueden ser usadas como *IP públicas* [46][47].

En la configuración, queda pendiente por cambiar el modo de direccionamiento de dinámico a estático. El modo dinámico significa que cada X tiempo nuestra IP cambia automáticamente y se asigna una IP aleatoria dentro del rango permitido. En modo estático o fijo elegiremos nosotros la IP que queremos y nunca cambiará, eso sí, tendremos que tener cuidado de que no existan solapamientos; en otras palabras, que ningún otro dispositivo use esa misma IP o se crearía un conflicto, lo que impediría su conexión a la red local. También necesitaremos conocer los servidores *DNS* de nuestro *router* para poder establecer el modo estático.

Los *DNS* (siglas en inglés de Domain Nomain Services; en castellano, Servidores de Nombres de Dominio) sirven para traducir nombres de dominio en direcciones IP que puedan comprender los diferentes dispositivos. Los servidores de *DNS* usan bases de datos para poder realizar esta traducción y contienen los nombres de todos los dominios de una red. Por ejemplo, en Internet los servidores de *DNS* deben conocer todos los nombres de los dominios a nivel mundial [48].

Como servidor de *DNS* podemos usar el que nos proporciona nuestro proveedor de *ISP*. Para conocer su *DNS* podemos realizar la consulta de dos formas distintas:

- Entrando en el *router* con un navegador desde la máquina anfitriona, escribiendo la siguiente url: **192.168.1.1**
- Consultado el fichero que se encuentra en la ruta: */etc/resolv.conf*

También podemos no usar el que nos proporciona nuestro proveedor de ISP y usar el que nos ofrece Google o cualquier otra empresa [49].

Quedándonos el archivo modificado como en la imagen que mostramos a continuación:

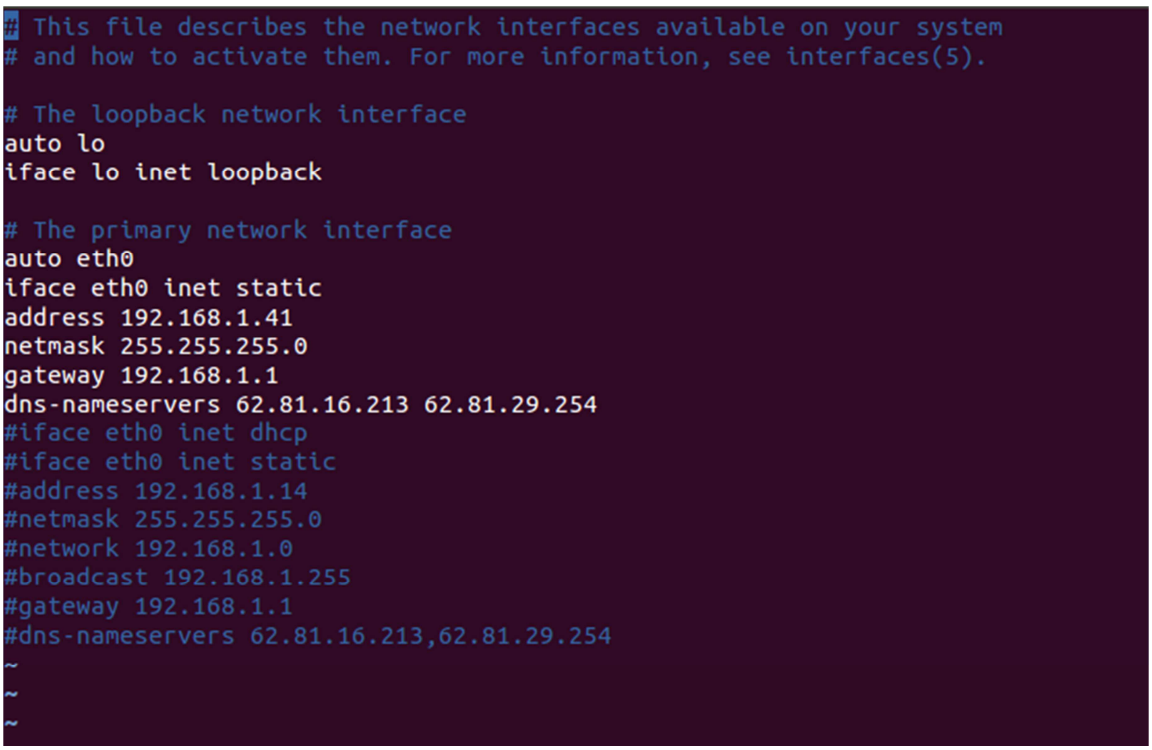


Figura 3.25 – Consulta de resolv.conf para averiguar el DNS

Por último, habilitaremos el escritorio remoto. El escritorio remoto permite que un usuario trabaje con un equipo sin estar enfrente de él. Esto se consigue realizando una conexión entre las dos máquinas a través de una red o de Internet. Esta funcionalidad nos será muy útil debido a que los tres integrantes del grupo podremos usar la máquina sin tener que estar físicamente delante de ella.

Uno de los requisitos para poder tener habilitado el escritorio remoto, aparte de activarlo, es que necesitamos abrir el puerto 5900 en el *router*. Un puerto es un canal de comunicación por el cual se pueden enviar y recibir datos. Ésta es una de las razones por las que necesitamos tener una *IP estática*, porque para abrir un puerto se le tiene que asociar la *IP privada* del equipo con el que se quiere realizar la transmisión de información.

❋ Arquitectura servidor de desarrollo

En la figura ilustramos cómo es la arquitectura del servidor de desarrollo, usando las tecnologías explicadas anteriormente:

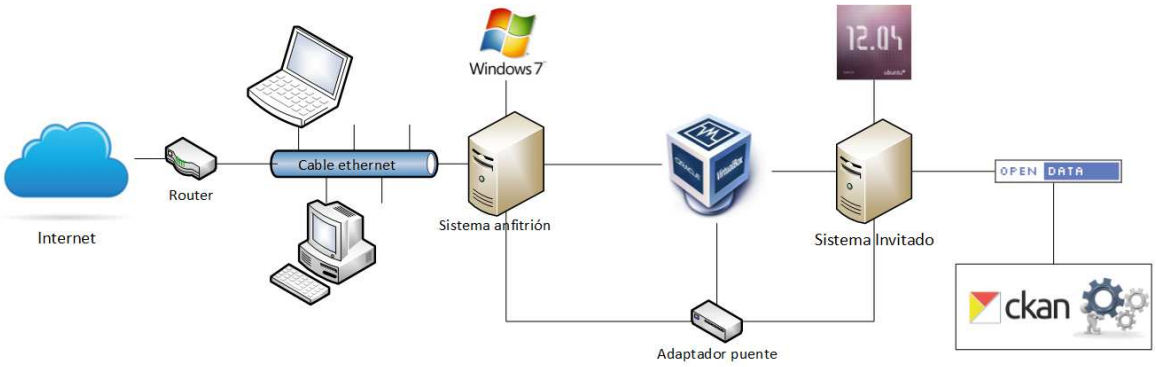


Figura 3.26 – Esquema de la arquitectura del servidor de desarrollo

3.3.3 Servidor de producción: Amazon Web Service

Como servidor de producción hemos decidido usar los servicios de *Amazon Web Service* debido a que nos ofrece una buena relación calidad - precio. Además, podremos ir aumentando y disminuyendo los recursos hardware necesarios a lo largo del tiempo sin ninguna dificultad.

En las siguientes secciones detallamos qué servicios usamos de *Amazon Web Service*, para montar nuestro servidor de producción.

❋ Amazon Web Service

Amazon Web Service (*AWS*) nos ofrece un conjunto de servicios de computación en la nube. La computación en la nube ofrece la posibilidad, a través de Internet y desde cualquier parte del mundo, utilizar algún dispositivo electrónico, fijo o móvil, con el cual podamos hacer uso de los servicios de computación que ofrecen diferentes proveedores, sin tener que

disponer físicamente de las máquinas necesarias para ello y solo pagando un cantidad de dinero por los servicios proporcionados.

Figura 3.27 – Logotipo de Amazon Web Services

Una de las grandes ventajas de la computación en la nube es que no necesitas disponer de los equipos, esto permite aprovechar mejor los diferentes recursos económicos. Otra gran ventaja es que permite escalar dinámicamente la potencia de los servicios de computación, dependiendo de la demanda puntual en ese momento. Esto también hará que optimicemos los recursos económicos debido a que sólo pagaremos por lo que necesitemos en ese momento [50].

Por tanto, elegimos usar la computación en la nube porque con muy pocos recursos económicos podemos disponer de un servidor. Los diferentes servicios en la nube se dividen en tres capas, que explicaremos a continuación desde el nivel más bajo al más alto [51].

- En el nivel inferior se encuentran los **Servicios de Infraestructura** (*IaaS*). En este nivel se sitúan los recursos físicos o hardware, es decir los servidores, el sistema de almacenamiento, etc., que se ofertan al cliente para su uso. Por norma general estos recursos se administran gracias a la virtualización.

- En la capa de en medio se encuentran los **Servicios de Plataforma** (*PaaS*), con lo que en este nivel podemos observar que ya no tenemos una máquina desnuda, sino que está instalado el sistema operativo, las bases de datos configuradas y tenemos preconfigurado un entorno de programación. Es decir, ya tenemos instalado el software necesario para desplegar la aplicación que estemos desarrollando. Ofertan al cliente un entorno en el que puede empezar a desarrollar su aplicación olvidándose de gestionar la infraestructura hardware y software de los equipos.

- En la capa alta se encuentran los **Servicios de Aplicaciones** (*SaaS*). En esta capa se están ya las aplicaciones listas para ofrecerlas como servicios, pudiendo ser

estas de pago o gratuitas. Para que lo entendamos mejor, un ejemplo sería un servicio de correo electrónico.

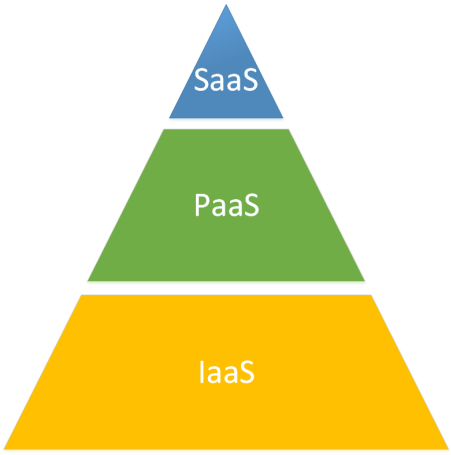


Figura 3.28 – Esquema de capas usado en Amazon Web Services

Podemos distinguir entre varios tipos de nube:

- **Nube pública:** Estas nubes están mantenidas y gestionadas por una empresa que ofrece sus servicios a terceros. Este tipo de servidores, aunque ofrecen al usuario una cuenta y un espacio personal, en realidad los datos alojados y los procesos de los diferentes clientes están entremezclados en los servidores compartidos, sin que los usuarios lo perciban.

- **Nube privada:** En este caso la organización es la que controla y gestiona la infraestructura y solo permite el acceso a un grupo de usuarios restringidos. Este tipo se suele usar cuando los datos que se manejan dentro de la infraestructura son muy confidenciales.

- **Nube híbrida:** Combina el uso de la nube privada y la pública.

- **Nube comunitaria:** En este tipo varias organizaciones comparten una misma infraestructura para un propósito común, puede estar gestionada por ellos o por terceros.

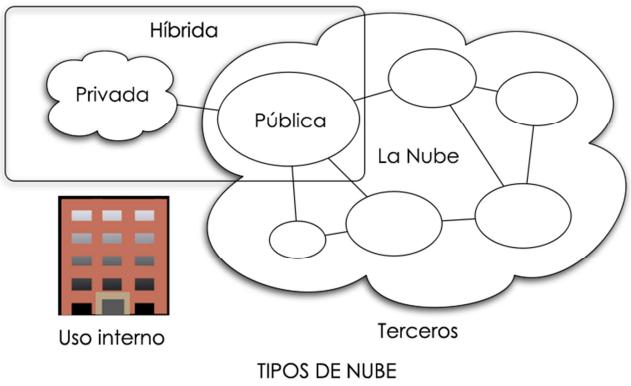


Figura 3.29 – Esquema de las diferentes computaciones en la nube existentes

Amazon Web Service es una nube pública y dependiendo de los servicios que se usen se encontrará en la capa de *IaaS* o *PaaS*.

❋ Amazon Elastic Compute Cloud

Hemos escogido *AWS* porque dentro de los servicios que oferta se encuentra *Amazon Elastic Compute Cloud* (*Amazon EC2*). Este servicio pertenece a la capa *IaaS*. *Amazon EC2* se basa en que el cliente elige un sistema operativo virtualizado diseñado para este propósito. Actualmente casi todos los sistemas operativos tienen versiones adaptadas y se instalan en una *Instancia* [52].

Una *Instancia* es un término que se define en *Amazon EC2* y significa que contratamos una cierta cantidad de hardware donde se instala nuestro sistema operativo.

Así pues, para usar *Amazon EC2,* lo primero que se tiene que hacer es contratar una *Instancia*. Para ello se tendrá que escoger entre una serie de *Instancias* predeterminadas donde se elegirán los recursos hardware que contendrá dicha *Instancia*. Estas *Instancias* se cobran por horas de funcionamiento y costarán más a mayor cantidad de recursos seleccionados, por lo que se deberían escoger sólo los que se vayan a usar y no contratar más de lo necesario.

En esta selección de configuración tendremos que decidir qué tipo de almacenamiento queremos usar: *Instance Store* o *Elastic Block Store* (*EBS*). La diferencia entre uno u otro es que en *EBS* el almacenamiento es permanente, es decir, aunque paremos la *Instancia* o la reiniciemos no perderemos la información contenida en ella, en cambio con la otra opción si perderemos la información, ya que ésta sólo durará mientras la *Instancia* esté en

funcionamiento. Además el almacenamiento *EBS* es más rápido, la única desventaja es que es considerablemente más caro [53].

A continuación se tendrá que elegir el sistema operativo que se quiera utilizar. En *Amazon* estos sistemas operativos se llaman *Amazon Machine Image* (*AMI*) que son los sistemas operativos adaptados para funcionar en los equipos de *Amazon EC2*.

Para nuestro servidor de producción queremos una *AMI* de *Ubuntu Server 12.04 64Bits*, esta se llama *ami-af7abed8* .

Una vez terminado este paso, la *Instancia* se lanzará y arrancará. Ahora podremos entrar en ella vía *ssh* y configurarla como queramos, pudiendo instalar todos los programas que se deseen siempre y cuando sean compatibles con esa versión del sistema operativo.

Una característica importante que vamos a explicar ahora son los llamados *Amazon EC2 Security Groups*. Un *Amazon EC2 Security Group* es un *firewall* en el que configuramos una serie de reglas, y lo asignamos a una *Instancia*. Toda *Instancia* tiene que tener asociado un *Amazon EC2 Security Group*. Con estas reglas indicaremos por qué puertos queremos que se permita tráfico, pudiendo ser este de entrada o de salida al exterior. En realidad es similar a abrir los puertos en un *router* [54].

Otro servicio que vamos a utilizar de *Amazon* son las *Amazon EC2 Elastic IP Address*. Este servicio sirve para asociar una *IP estática* a una *Instancia* y así poder asignarle un dominio, ya que por defecto las *Instancias* tienen una *IP dinámica*. Esto obedece a razones de seguridad para evitar hackers [55].

El último servicio que vamos a usar es el de crear una *AMI* de nuestra *Instancia*. Esto sirve para crear una copia de seguridad de la *Instancia* que estamos usando, ya que si por alguna razón debemos eliminarla, podremos volver a iniciar una nueva perfectamente configurada con solo un clic desde el panel de configuración de *Amazon*. Se creará cuando hayamos terminado de configurar la *Instancia*.

❋ Arquitectura servidor de producción

En el siguiente diagrama ilustramos cómo es la arquitectura del servidor de producción usando las tecnologías explicadas anteriormente:

Figura 3.30 – Esquema resumido de la arquitectura del servidor en producción

3.4 Arquitectura final Open Data

Para finalizar, vamos a resumir en un diagrama la arquitectura total del servidor de producción de nuestro Open Data junto con las tecnologías usadas en el mismo.

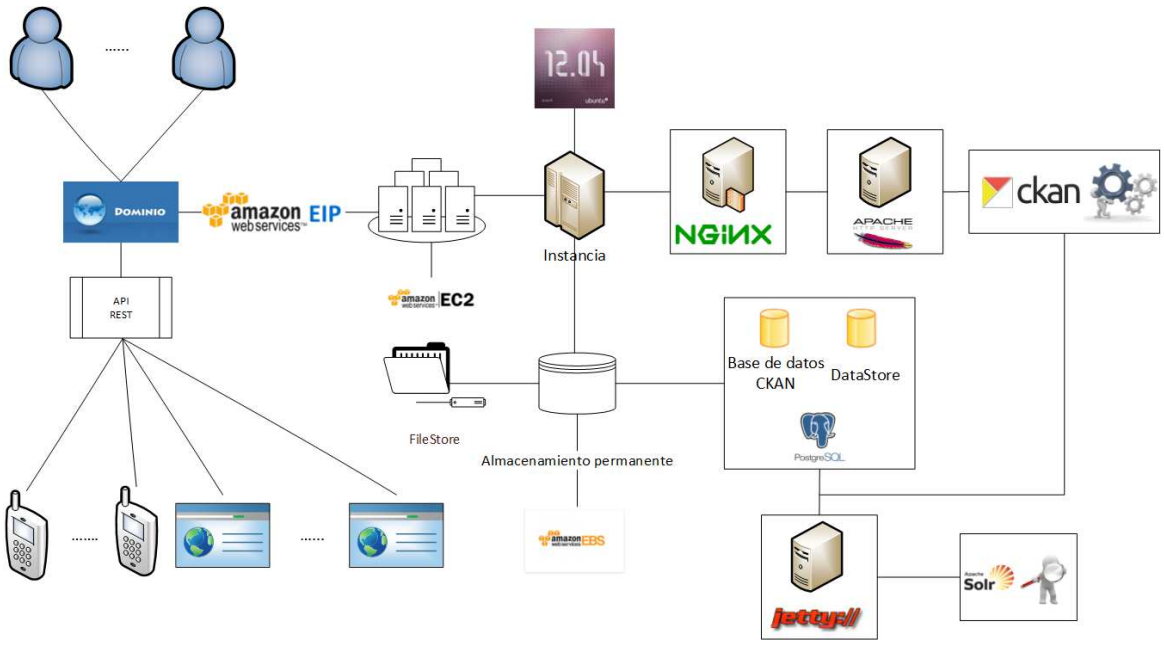


Figura 3.31 – Esquema de la arquitectura total del servidor en producción

IV. Aplicaciones de Medio Ambiente

Una vez hemos dejado atado el problema de cómo unificar la información y hacerla visible y accesible desde diversos procedimientos (la parte del servidor) podemos comenzar a ejecutar la segunda empresa de este proyecto, en la cual centraremos nuestra atención en las aplicaciones y oportunidades que puede brindarnos nuestro Sistema Open Data (la parte del cliente).

Como ya adelantamos en el capítulo anterior, comenzaremos trabajando en el problema de la integración y adaptación de las aplicaciones desarrolladas por el grupo G-Tec durante el curso 2012-2013. Utilizaremos este punto de partida como base en el tratamiento de los desafíos que se van a plantear en las siguientes páginas. Entre los nuevos temas a solucionar se encuentran: la identificación de cuáles serán los conjuntos de datos que al final coparán nuestro gestor de contenidos o, en qué tipos o formatos deberán estar definidos dichos conjuntos. Para este último punto facilitaremos el trabajando proponiendo una serie de directrices que los conjuntos de datos deberán aplicar para el correcto funcionamiento de los sistemas.

Por último, hablaremos acerca del protocolo de comunicación entre los usuarios y el gestor de contenidos. En este apartado final explicaremos como se conectan las apps al Open Data de manera particular y acabaremos dando detalles sobre cómo realizar las consultas y usar los datos de manera general.

4.1 Integración de nuestras apps como sistemas Open Data

El proceso central de la integración de las aplicaciones de G-Tec, comprende la tarea de cambiar la estrategia seguida hasta ahora para comunicarse con el servidor, por otra nueva que es la que exige nuestro sistema Open Data. Una vez hecho esto, todas las modificaciones se deberán a razones particulares de cómo trata cada aplicación la información internamente o a requisitos adicionales demandados por el Ayuntamiento.

Durante las siguientes secciones nos gustaría tratar las pautas o pasos que habría que seguir en cada una de las seis aplicaciones de G-Tec para lograr su completa inmersión en el Open Data, pero tal trabajo comprendería bastante más de un año de proyecto. Esto es debido a las distintas arquitecturas internas de diseño que se han utilizado en las apps, por lo que hemos decidido ser realistas y escoger una aplicación para empezar y obtener unos primeros resultados que permitan allanar el camino a las demás. No obstante, aunque las ideas mostradas a continuación estén enfocadas hacia una solución particular de una

aplicación concreta, procuraremos facilitar la tarea a la hora de extraer las ideas generales en los pasos seguidos, de manera que trasladar estas directrices al resto del conjunto de apps no suponga mucha dificultad.

Tal y como anunciamos en el capítulo anterior, nos decantamos por la aplicacion Mapa de Recursos Ambientales como punto de inicio. Una de las razones principales se debe, como ya entonces mencionamos, a que esta app era la que más se acercaba a satisfacer los estándares de diseño que dicta el IAM. Esto hace de Mapa de Recursos Ambientales la candidata perfecta para pasar las pruebas que le permitiría empezar a formar parte del catálogo de aplicaciones disponibles para los ciudadanos de Madrid.

En este punto, queremos crear un inciso para aclarar que, cuando hablamos de cumplir los estándares del IAM, estamos refiriéndonos sobre todo al objeto que actúa como servidor de estas aplicaciones móviles, puesto que los programas clientes se desarrollan en Java nativo (Android) y no crean ningún conflicto. Es en el diseño del servidor donde más divergen todas las apps y sobre todo Mapa de Recursos Ambientales. Mientras que en las demás se han realizado implementaciones sencillas usando el lenguaje PHP (algo totalmente prohibido en proyectos del IAM) tanto para los servidores como para los casos que incluyen servicios web, en Mapa de Recursos se ha desarrollado una aplicación web JEE (*Java Enterprise Edition*; en castellano: Java Edición Empresarial) siguiendo estrictamente los cánones del IAM para representar al servidor y su base de datos.

Volviendo a las razones que nos empujaron a realizar la integración de esta app, destacamos también que el tratamiento de los datos se realiza en un único sentido; esto es, la aplicación solo hace lecturas a la base de datos sin producir modificaciones. Este hecho permitirá que las primeras pruebas con el sistema sean más sencillas de manera que la depuración no conlleve una cantidad excesiva de trabajo.

Por último, podríamos considerar a Mapa de Recursos como la aplicación más compleja en cuanto a implementación se refiere. Debido a su diseño estructurado en capas, es necesario que se incorporen varias librerías y *frameworks* (tanto en el servidor como en el cliente) para su funcionamiento que el resto de apps no necesitan. Si tenemos en cuenta que nuestro sistema simplifica bastante este tipo de cosas al no precisar ningún software adicional para realizar las conexiones y recuperar datos desde el gestor de contenidos, resulta interesante empezar tratando esta aplicación. De esta manera, comprobaremos que el no haber utilizado el modelo de capas en el resto de aplicaciones va a resultar casi más favorable para la integración, pues el paso en el que retiramos las librerías innecesarias con su correspondiente limpieza de código, no se habrá de realizar en ellas.

4.2 Migrando Mapa de Recursos Ambientales al Open Data

Antes de proceder con la adaptación de la aplicación, consideramos útil e interesante realizar un breve análisis acerca del funcionamiento y la estructura interna inicial que ayuden a comprender de dónde partimos y hacia dónde queremos dirigirnos.

Como ya se ha venido diciendo desde el capítulo tres, Mapa de Recursos Ambientales [56] sigue una implementación basada en capas que da forma a todas las infraestructuras implicadas para representar el proyecto. Esto permite aislar unas funcionalidades de otras al delimitar claramente de que se ocupará cada capa. El esquema que resume esta situación es:

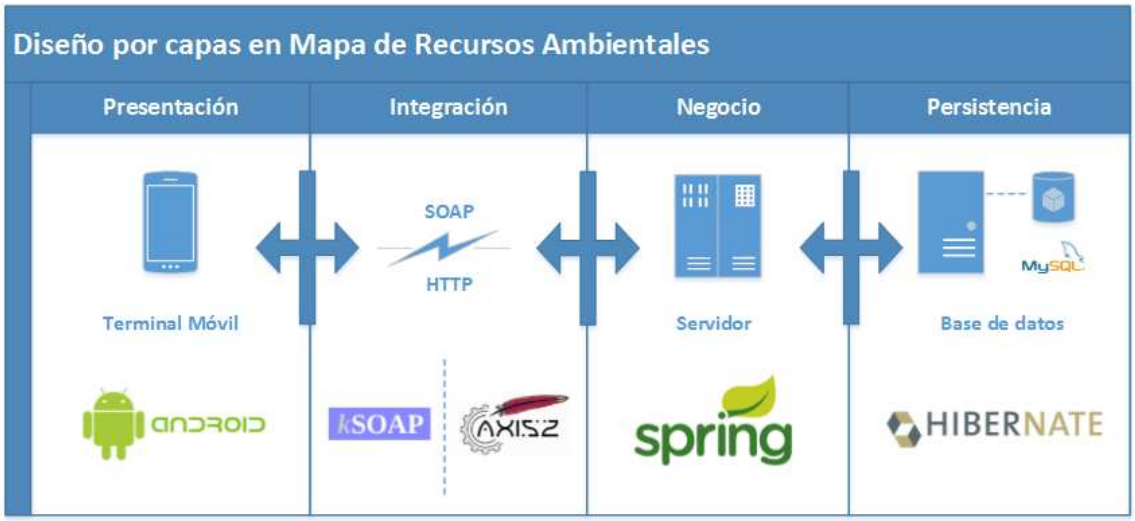


Figura 4.1– Esquema de la arquitectura por capas de Mapa de Recursos 1.0

En la figura se pueden observar tres capas principales, las cuales se definen a grandes rasgos como:

- **Capa de persistencia:** Es la capa que se encarga del acceso a los datos. En este caso se utilizó el *framework Hibernate* que permite tratar y modelar registros de una base de datos como si fueran objetos Java.

- **Capa de negocio:** Es la capa que administra la lógica de negocio, o lo que es lo mismo, el funcionamiento interno del servidor, ya que esta se sitúa como enlace entre las demás facilitando la comunicación entre ellas. El hecho de usar *Spring* aquí simplifica el diseño, ya que las implementaciones de los objetos se basan en interfaces que en tiempo de ejecución adoptarán instancias concretas, lo que redunda en una reducción del código a escribir.

- **Capa de presentación**: Esta capa pertenece a la parte del cliente, y es la encargada de mostrar la aplicación final en la interfaz de usuario. En este caso son los terminales móviles con el sistema operativo Android.

La característica principal de este modelo es que se consigue una centralización de la lógica de negocio compactándola en una capa únicamente. Así, la capa de presentación nunca operará directamente sobre el gestor de la base de datos o incluso sobre la misma base de datos, sino que le pedirá a la capa de negocio que lo haga por ella y viceversa.

Además de estas capas, en la figura podemos apreciar una más que es la que sirve de puente entre las capas de presentación y de negocio. Se trata de la capa de integración que es la que implementa el servicio web, el cual sirve de canal de comunicación entre los terminales móviles y el servidor.

El intercambio de mensajes se lleva a cabo mediante peticiones SOAP (siglas de *Simple Object Access Protocol)*, que no es más que un protocolo para el intercambio de datos en formato XML con una estructura prefijada. El encargado de recibir dichas peticiones y decirle a la capa de negocio qué servicio se ha solicitado es el framework *Axis2*. Esto es debido a que cuando usamos el protocolo SOAP, realizamos la comunicación con el servidor siempre a través de la misma URL, por lo que hay que indicarle qué tipo de acción se requiere. Eso se hace encapsulando la petición concreta y sus parámetros en un mensaje SOAP. Para crear dichas peticiones y decodificar las respuestas del servidor se inserta en la app la librería *ksoap2* que lleva a cabo este proceso de codificación/decodificación.

Ahora que ya tenemos una idea clara de la estructura de la aplicación, podemos hacer un resumen de su funcionamiento. Para ello, vamos a mostrar un caso de uso muy básico que ejemplifica una petición del usuario al servidor y su correspondiente respuesta. Nos centramos en dicho caso porque es el más relevante en nuestro proceso de integración:

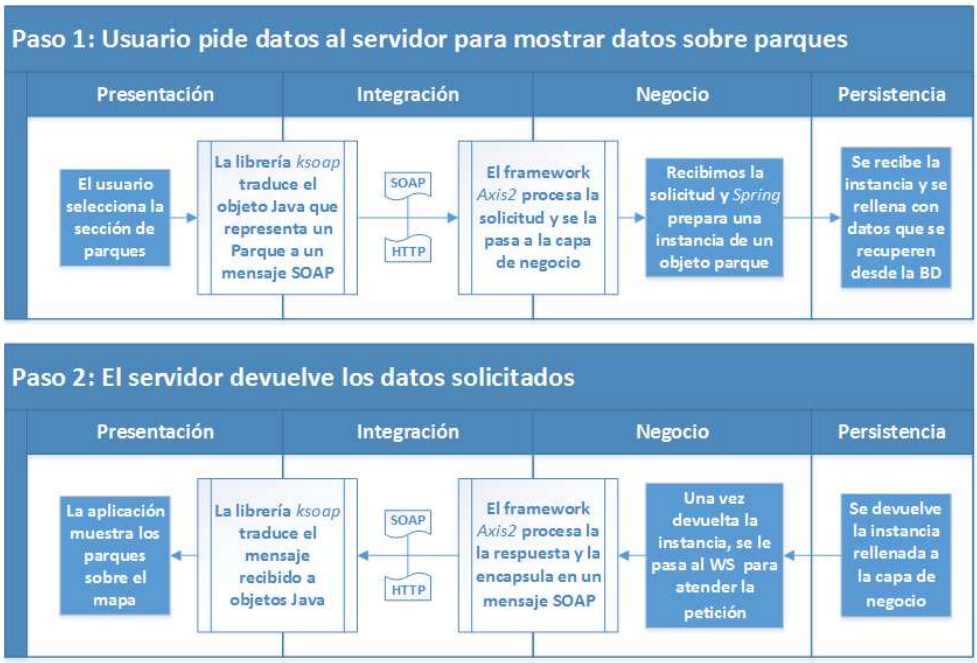


Figura 4.2– Caso de uso: Petición al servidor en Mapa de Recursos 1.0

Con todo esto, ya podemos hacernos una idea clara de cómo es la aplicación Mapa de Recursos de la que partimos. A partir de este momento, podemos empezar a explicar cuáles van a ser los cambios necesarios en dicha aplicación para que empiece a tomar los datos desde el sistema Open Data. Lo primero que haremos será mostrar unos diagramas análogos a los anteriores en el que se aprecien claramente las diferencias acerca de cómo se van a realizar ahora las consultas:

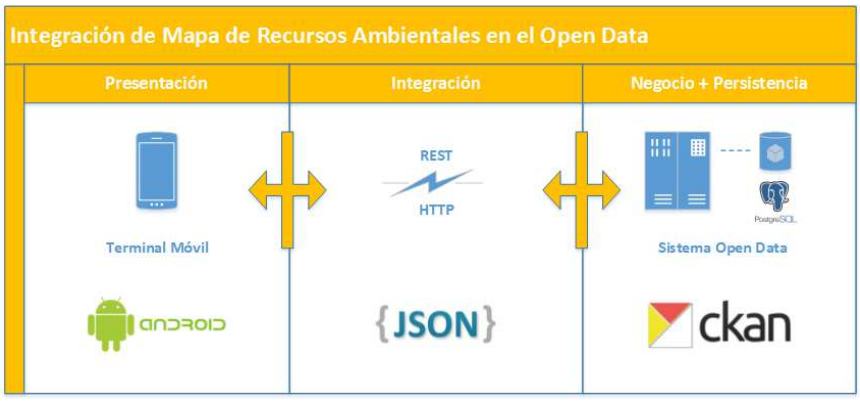


Figura 4.3 - Esquema de la arquitectura por capas de Mapa de Recursos 2.0

Hay que hacer notar, que ahora debemos enfocar estas aplicaciones (y cualquiera que se desee realizar) sólo desde la parte del cliente, dado que los individuos o entidades que decidan emprender un proyecto utilizando datos del Sistema Open Data no estarán en posición de administrar dichos datos ni el propio sistema. Dicha gestión pasa a ser tarea del equipo administrador del Sistema Open Data (actualmente nosotros, y en un futuro, el Ayuntamiento de Madrid).

Lo único que se les permite a los usuarios es visualizar los datos, descargar la copia más reciente y hacer consultas (también modificaciones en casos aislados). Por lo tanto, con la solución Open Data hacemos que los desarrolladores se puedan concentrar exclusivamente en implementar los programas cliente, permitiendo que se olviden de los detalles que se esconden tras el servidor de datos y su funcionamiento; en definitiva, sus capas. De hecho, con esta solución, al programador le es suficiente con solo conocer la API de consultas para el conjunto de datos que desea utilizar y su correspondiente URL para lanzar las peticiones HTTP.

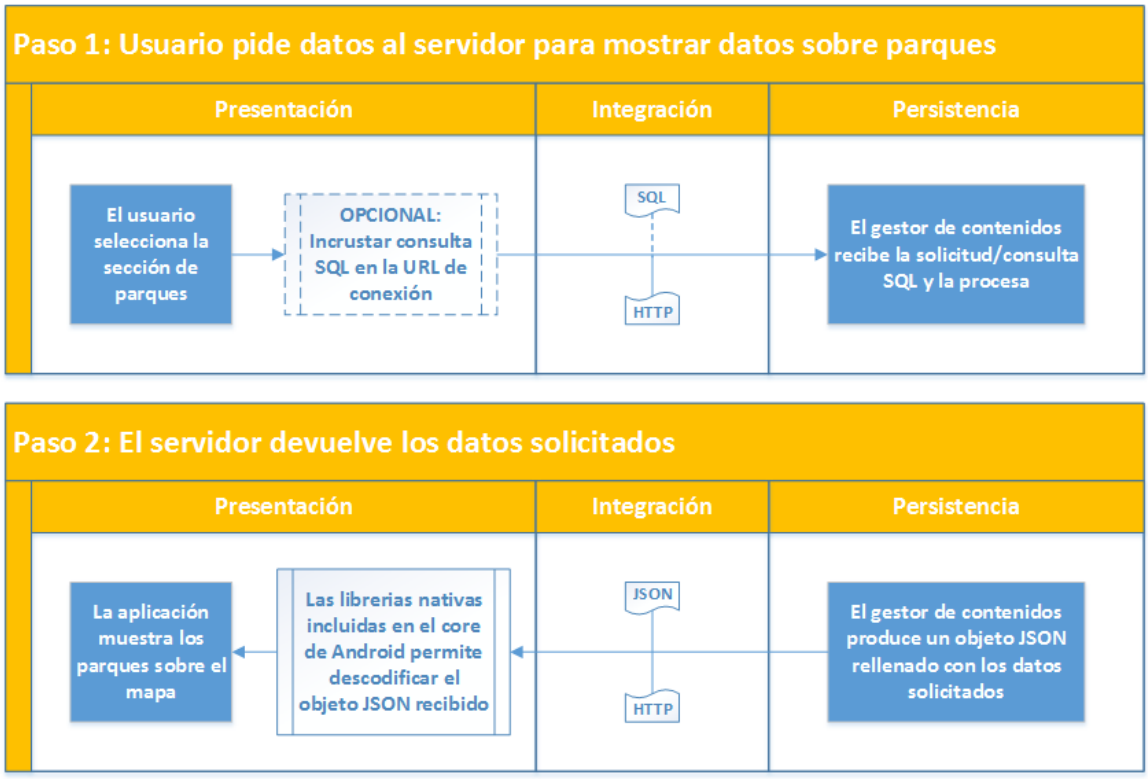


Figura 4.4 - Caso de uso: Petición al servidor en Mapa de Recursos 2.0

La razón de esta sencillez se debe al cambio de paradigma: Del protocolo SOAP a una API REST. A diferencia del caso anterior, ahora no hay una única URL a la que enviar

peticiones, sino que hay conjuntos (la API de llamadas) que hacen referencia a acciones o recursos en el servidor. Además, los mensajes intercambiados tendrán formato JSON en lugar de estructura SOAP para los cuales no es necesario incluir ninguna librería en el cliente que los interprete (viene incluida en el *core* de Java/Android).

Todos los detalles de conexión los trataremos con más tranquilidad en la última sección de este capítulo dedicada a los **Protocolos de Comunicación**. Por el momento, debemos quedarnos con la **idea general** de que es necesario reestructurar el código de las aplicaciones para conectarse con el Open Data. A grandes rasgos:

- **Se debe retirar cualquier librería incluida por necesidad de comunicación con el servidor.** Aquí nos referimos a casos como ksoap2, que ya no van a ser necesarias.

- **Hay que modificar las partes del código o métodos que se dediquen a encapsular las peticiones al servidor**. Por ejemplo, si antes preparábamos un mensaje SOAP, ahora tan solo debemos construir una URL en casos de lectura y una URL y un objeto JSON en casos de escritura. Las conexiones seguirán realizándose con el protocolo HTTP.

- **Hay que modificar las partes del código o métodos que se dediquen a encapsular las respuestas del servidor**. Siguiendo el mismo ejemplo, si antes decodificábamos un mensaje SOAP como respuesta, ahora deberemos decodificar un objeto JSON. El método de captura (protocolo HTTP) de dicho objeto es el mismo que para SOAP, tan solo cambia el contenido de los datos recibidos y su procesamiento.

Hasta aquí, podemos concluir a grosso modo lo que denominábamos el proceso central de la integración (salvando los detalles del protocolo de comunicación, que como dijimos, se darán más adelante). Consideramos que dicho proceso debería ser más sencillo de aplicar en aquellas apps de G-Tec que no implementen el modelo por capas. Por lo tanto, desde este momento empezaremos a sopesar problemas particulares de Mapa de Recursos Ambientales que concluyan con su adaptación. No obstante, volvemos a insistir que algunas de las ideas que se van a exponer se podrían considerar y aplicar al resto de aplicacioness.

4.2.1 Problemas adicionales

Además de adaptar el sistema de conexión con el servidor, era necesario realizar algunos retoques más en la app:

- **Eliminación del proceso de Inicio de Sesión y el sistema de usuarios.**

  En la versión inicial de la aplicación se exigía el registro del usuario para poder acceder a los servicios que ofrece la aplicación. Se nos sugirió retirarlo durante alguna de las reuniones celebradas con el ayuntamiento. El hecho de incluir registro de usuario estaba orientado a meros fines estadísticos para averiguar qué perfil de usuario utilizaba la aplicación. Por otra parte, tiene sentido que si en el Sistema Open Data no requerimos ningún registro para consultar los datos, tampoco se haga en la app. En el resto de aplicaciones, este punto sería importante considerarlo y valorar la necesidad real de disponer de perfiles de usuario.

- **Eliminar cargas locales de los recursos.**

  Los datos relativos de algunos de los recursos, como las Áreas de Prioridad Residencial, se encontraban almacenadas de manera local en la aplicación. A partir de ahora estos datos, y los de todos los recursos, se deberán cargar con información proveniente del Sistema Open Data, de manera que se asegure la obtención de datos actualizados en todo momento. Para el resto de apps, si la información a cargar de manera local está disponible en el Sistema Open Data, este último debería tener prioridad como fuente de datos.

- **Actualización de los datos de los recursos.**

  Para la implementación de la aplicación, el Ayuntamiento proporcionó los ficheros que contenían los datos de los diferentes recursos a mostrar. Dado que la aplicación se implementó en el curso 2012-2013, la información disponible entonces se podría considerar ahora obsoleta, pues muy probablemente, esta haya sufrido cambios. Estas cuestiones las trataremos en la sección de **Normalización de los datasets** con más detalle. Las demás apps deberían revisar sus conjuntos de datos y pensar si sería conveniente solicitar una copia actualizada al ayuntamiento.

- **Cambios en la forma de mostrar los recursos.**

  Continuando un poco con el punto anterior, los archivos (conjuntos de datos) proporcionados este año no son necesariamente iguales a los que ya se tenía. Esto ha forzado a realizar algunos cambios dentro de la aplicación, como eliminar el filtro para instalaciones disponibles dentro de un parque, o la inclusión de más recursos que en la primera versión no existían.

.

Teniendo en cuenta sobre todo los dos primeros puntos (además de adaptar la conexión) y el código fuente del que partíamos, nos parecía más razonable reescribir la aplicación de cero que ir modificando lo que ya teníamos. Además, podíamos aprovechar la ocasión para cambiar el rango de versiones de Android compatibles con la aplicación. Acerca de esta última iniciativa hablaremos brevemente en la siguiente sección.

4.2.2 El fenómeno de la fragmentación en Android

Hemos querido crear este inciso para explicar un problema al que se enfrentan todos los programadores que deciden emprender proyectos en Android. Se trata de un hecho conocido como *la fragmentación* de Android, y debe su nombre a la multitud de versiones del sistema operativo, cuyos ciclos de vida, se encuentran todos activos a la vez. Esto provoca, que antes de empezar a realizar un diseño, se deba decidir cuánta *retrocompatibilidad* queremos ofrecer. Para ayudarnos a tomar esta decisión, existe un apartado en la página de desarrolladores de Android denominado ***Dashboards***, donde se muestran estadísticas globales que se actualizan regularmente [57].

A medida que aumentamos la compatibilidad con versiones más antiguas, logramos que nuestro radio de alcance aumente, ya que la app está disponible para más terminales. Como contrapartida, la labor de programación puede volverse más tediosa al forzar al programador a implementar manualmente características que solo están disponibles en las últimas versiones. Google intenta remediar esto proporcionando algunas librerías de soporte que permiten que estén disponibles características de las últimas versiones en las más antiguas, pero esto no cubre todos los casos.

Por otro lado, si reducimos el rango de versiones que podrían instalar nuestra aplicación, perdemos cuota de mercado a cambio de facilitar la programación. A esto debemos añadirle que también contribuimos al mantenimiento del código gracias a la simplificación ganada, además de evitar los errores en ejecución que puedan surgir debido a los procesos de adaptación; cabe esperar que algo diseñado para las últimas versiones, pudiera llegar a fallar para las más antiguas.

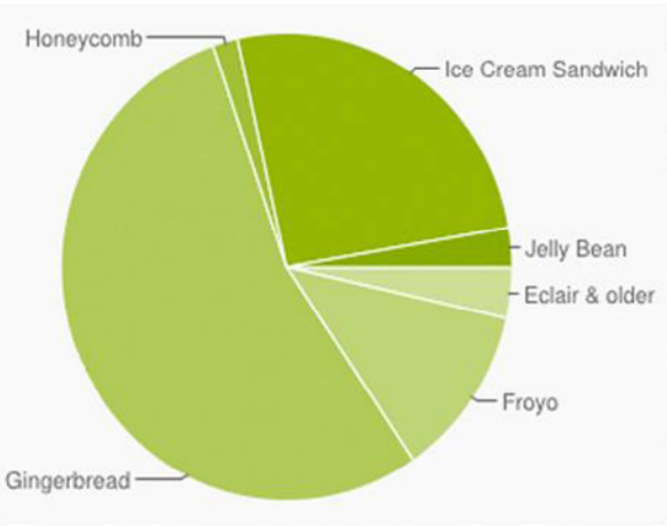
Google intenta evitar la fragmentación a toda costa, pues perjudica los desarrollos y suele ser el origen de muchas de las frustraciones que los programadores de aplicaciones para Android sufren, aunque hay que remarcar, que no es empresa sencilla de llevar a cabo; requiere del consenso de muchas partes. Entre algunas de las medidas que Google ha emprendido para paliar el problema, se encuentran [58]:

- Durante el Google/IO 2011 [59] anunció que había llegado a acuerdos con los socios que fabrican dispositivos que utilizan Android: Estos últimos se comprometían a actualizar a la última versión, al menos, por un periodo de 18 meses desde el lanzamiento del terminal.

- Durante el Google/IO 2012 [60] añadió otra propuesta más que consiste en adelantarles en primicia las nuevas versiones a los socios para que no se demoren tanto en lanzar sus actualizaciones. Se intenta evitar que algunas marcas actualicen hasta tres meses después de que Google lance su actualización oficial.

- Por último, y la que parece que da mejores resultados al depender sólo de Google, consiste en independizar algunas aplicaciones del sistema del propio núcleo. De esta manera, es posible disfrutar de la última versión de las aplicaciones oficiales de Google (Maps, Youtube, etc) sin necesidad de actualizar la versión de Android.

Para poner cifras concretas a la distribución de cuotas y su evolución en los últimos dos años (2012-2014), vamos a comenzar por Noviembre de 2012, momento en el que se inició el desarrollo de la app Mapa de Recursos Ambientales. Entonces los datos revelaban lo siguiente:

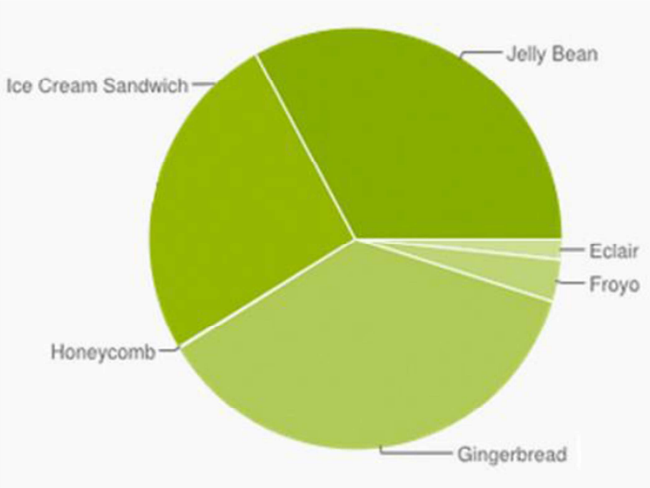


Version	Codename	API	Distribution
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Eclair	7	3.1%
2.2	Froyo	8	12%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	53.9%
3.1	Honeycomb	12	0.4%
3.2		13	1.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.8%
4.1	Jelly Bean	16	2.7%

Figura 4.5 – Dashboard *correspondientes a Noviembre 2012*

Por entonces, se decidió escoger como versión mínima la que casi todos los desarrolladores suelen escoger: la versión 2.3.3 (*Gingerbread* - API 10). De esta manera, conseguían alcanzar al 84.5% de los terminales activos.

En el siguiente gráfico podemos ver recogidos estos datos de nuevo, pero esta vez en el momento que se finalizó la implementación de la aplicación (Junio de 2013). Se puede observar el lanzamiento de una nueva versión al mercado (Android 4.2 - *Jelly Bean* - API 17), lo que les obligó a actualizar el compilador si se deseaba inscribir a los dispositivos con esa última versión dentro del dominio de alcance. El valor final de dicho dominio se fija en el 95% aproximadamente.



Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.5%
2.2	Froyo	8	3.2%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	36.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.6%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	4.0%

Figura 4.6 – Dashboard *correspondientes a Junio 2013*

En la figura 4.6 podemos observar la primera caída de cuota de la versión 2.3 (*Gingerbread*), en aproximadamente 7 meses pierde el 18% de los dispositivos. Esto puede deberse a que los usuarios renovaron el terminal, actualizaron la versión o se pasaron a un dispositivo con un sistema operativo distinto a Android.

Al comienzo del presente proyecto (Octubre 2013) volvimos a consultar los *dashboards*. Comprobamos que de nuevo, la versión *Gingerbread* había perdido usuarios; concretamente abarcaba un 25% (un 11% menos que la vez anterior). Dada esta tendencia a la baja, decidimos entonces dejar de dar soporte a esta versión y reescribir la app para dispositivos que ejecuten como mínimo la 4.0 (*Ice Cream Sandwich*).

La siguiente figura contiene datos relativos a Junio de 2014:

Version	Codename	API	Distribution
2.2	Froyo	8	0.8%
2.3.3 - 2.3.7	Gingerbread	10	14.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	12.3%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	19.1%
4.3		18	10.3%
4.4	KitKat	19	13.6%

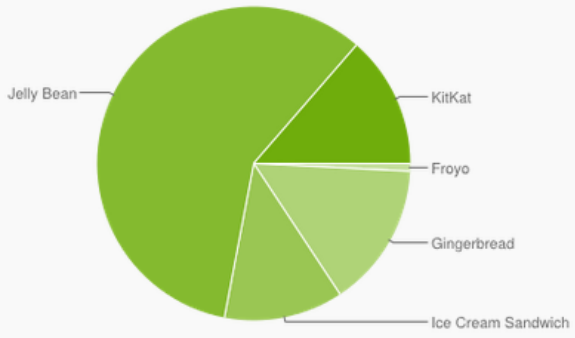


Figura 4.7 – Dashboard *correspondientes a Junio 2014*

Aquí se demuestra la buena elección que hicimos al subir la versión mínima: *Gingerbread* refleja un 10% adicional en pérdida. La nueva versión implementada acapara aproximadamente el 85% de los terminales y cabe pensar, a la vista de los acontecimientos observados, que este número aumentará en un futuro no muy lejano. Por último destacar la desaparición de la versión 3.X (*Honeycomb*) de Android, debido a que Google no muestra las versiones con menos de un 0.1% de uso.

4.2.3 Mapa de Recursos Ambientales: Versión 2.0

Una vez tenemos claro que vamos a reescribir el proyecto y cuál será la versión mínima del sistema operativo que los terminales deben ejecutar (recordamos, la 4.0), podemos proceder con la adaptación. Antes de comenzar y llegados a este punto, parece natural que definamos esta nueva aplicación como una nueva versión de lo que ya existía, así con *Versión 1.0* haremos referencia a la implementación original de la aplicación, mientras que con *Versión 2.0* estaremos hablando de la reedición que utiliza el sistema Open Data.

Consideramos la Versión 2.0, precisamente como una actualización de la Versión 1.0, al tratarse de la misma aplicación con mejoras y nuevos recursos para mostrar al usuario. En general se ha podido reutilizar poco código, dado que eran necesarios cambios en las estructura de clases: algunas se han retirado (como las que hacían referencia al perfil de usuario que ya no necesitamos) y otras se han compactado (como las conexiones al servidor). Lo que sí podemos asegurar con certeza, es que las ideas y la filosofía introducidas en la versión 1.0 se han mantenido en esta nueva edición y que, como mucho, se han intentado mejorar. Mostramos estos detalles en el siguiente apartado.

Para el proceso de adaptación hemos utilizado las siguientes herramientas, por lo que es posible que en algunos momentos se haga referencia a ellas durante las siguientes secciones:

- **Eclipse Standard/SDK** (Version*: Kepler Service Release 2*)
- **Android SDK** (Revision 22.6.4)
- **Android ADT Plug-in** para la integración del SDK de Android dentro de Eclipse. Se puede obtener a través del Eclipse Marketplace (en el menú de Ayuda).

❋ Arquitectura del cliente

Resumimos hasta el momento las características que ya hemos decidido para la reedición de la app:

- **Sistema operativo:** Android
- **Versión del compilador:** Android 4.4.2 (*KitKat*) [API 19]
- **Versión mínima requerida:** Android 4.0 (*Ice Cream Sandwich*) [API 14]

Cualquier proyecto que se desarrolle en Android debe constar como mínimo de una serie de elementos que revisaremos detalladamente a continuación:

- Un descriptor de la aplicación (AndroidManifest.xml).
- Una carpeta (**src**) donde almacenar el código fuente Java.
- Una carpeta (**res**) donde almacenar el resto de recursos (imágenes, ficheros, descriptores de capa (XML), etc)

Usualmente, la programación en Android está orientada a Actividades. Estas se podrían definir coloquialmente como cada *pantalla* que se muestra al cliente; es decir, es una colección de vistas que todas ellas juntas forman la interfaz gráfica, pero que se van intercambiando según la necesidad. Por poner un ejemplo cercano, en nuestra aplicación, el menú principal es una actividad. Cuando se pulsa en alguna sección se lanza otra actividad mostrando el siguiente menú de opciones, y así sucesivamente hasta llegar a la actividad que muestra el mapa.

Las actividades se suelen definir como dos archivos con el mismo nombre y distinta extensión. Por ejemplo:

- **MiActividad.xml:** En este fichero se suelen declarar los elementos que compondrán la vista final, como los campos de texto, los botones, imágenes, así como la estructura de la vista y la posición de los elementos dentro de ella.

- **MiActividad.java:** En este fichero es donde se programa la funcionalidad de los elementos declarados en el fichero XML. Si declaramos un botón, en esta clase podremos crear un método que incluya el código a ejecutar en caso de ser pulsado.

Resumiendo, el XML suele describir la apariencia de la actividad y en la clase Java se define su funcionalidad; esto es lo que se conoce como *método declarativo*. Decimos *suele*, porque es posible incluir funcionalidad en los XML así como declarar nuevos elementos en la clase Java. Hacerlo de este último modo es lo que conoce como *programatically*, que se podría entender como *hacerlo de manera dinámica* o procedimental.

Hemos hecho un breve resumen de las actividades en Android para ayudar a comprender la estructura de directorios que vamos a presentar a continuación, que es la que sigue el proyecto en la parte del cliente. La siguiente imagen resume la estructura:

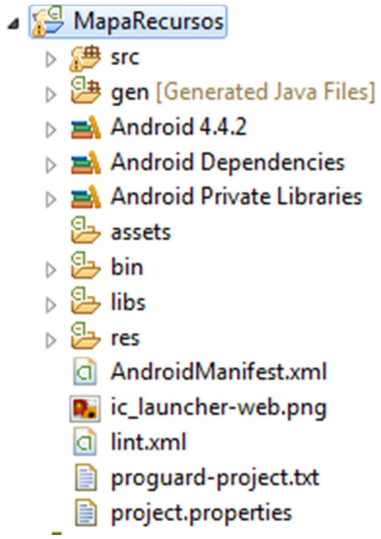


Figura 4.8 – Esquema resumen de la arquitectura del cliente

- **src:** Directorio que contiene el código fuente en ficheros Java. Analizaremos su contenido más adelante.

- **gen:** Este es uno de los directorios donde el SDK almacena los archivos, generados automáticamente, que usará para la compilación. El contenido de esta carpeta nunca debe ser modificado manualmente por el usuario si no se quiere incurrir en errores difíciles de solucionar. Dentro se encuentran 2 paquetes (uno de nuestra aplicación y otro de la parte de los mapas) que contienen los siguientes ficheros:

    - **BuildConfig.java:** Contiene una constante que define si la aplicación se encuentra en fase de desarrollo o en producción. Esta clase solo se encuentra en el paquete de nuestra aplicación dado que los mapas se consideran un plug-in.

    - **R.java:** Clase que contiene una relación con los identificadores de todos los recursos declarados en la aplicación, asociado cada uno a su dirección de memoria. Cada vez que definimos un elemento (o lo eliminamos) dentro de un XML y le adjudicamos un identificador, Eclipse automáticamente actualiza **R.java** para que después podamos hacer referencia a dicho elemento desde el código Java.

- **Android 4.4.2:** Es la librería que contiene el *core* de Android.

- **Android Dependencies:** No es un directorio. Es un resumen del conjunto de librerías oficiales, actualmente enlazadas al proyecto, y que son necesarias para su funcionamiento. En nuestro caso solo se encuentra:

    - **google-play-services-lib.jar:** Necesaria para poder mostrar mapas de Google dentro de la aplicación.

- **Android Private Dependencies:** Similar al anterior, pero resume el conjunto de librerías oficiales adicionales necesarias en esta aplicación en concreto. Aquí están situadas:

    - **google-play-services.jar:** Necesaria para poder mostrar mapas de Google dentro de la aplicación.

- **assets:** No ha sido usada. En esta carpeta se podrían incluir todos los ficheros que se quieran utilizar en la aplicación. Es similar a la carpeta *res*, salvo que ni se modifican ni se referencian los elementos en el *R.java*.

- **bin:** Es donde se genera el código compilado. Dentro se encuentra el fichero *MapaRecursos.apk* que sería la app lista para instalar en los dispositivos.

- **libs:** Es la carpeta física que contiene las librerías del proyecto. Después se deben enlazar para que aparezcan en ***Android Private Libraries.*** Aquí solo

encontramos la librería de soporte, ya que la de los servicios de Google Play se añade como un proyecto a parte.

- **res:** Aquí es donde realmente se suelen insertar todos los recursos necesarios para la aplicación, como imágenes, capas XML, etc. Como hemos dicho anteriormente, para todo lo que se incluye en este directorio, Eclipse crea una referencia a ello automáticamente en el R.java. No obstante, debido a todos los formatos de ficheros que pueden encontrarse aquí, se ha dividido en subdirectorios.

    - **drawable:** Este contiene todos los archivos de imagen (JPG, PNG, GIF, etc.) así como descriptores de imágenes en XML (figuras simples que se dibujan a partir de código XML). Aquí se encontrarían los gráficos que la app usa por defecto.

    - **drawable-hdpi, drawable-ldpi, etc:** Es igual que la anterior pero aquí se sitúan, opcionalmente, copias de los archivos gráficos orientados a diferentes densidades de pantalla (hdpi: Alta densidad; ldpi: Baja densidad, etc.).

    - **layout:** Contiene los XML que definen las vistas de las actividades que describimos anteriormente.

    - **menu:** Es parecida al anterior, y se usa solo en caso de incrustar una *ActionBar* en la aplicación. Aquí se definen los botones y menús de la barra.

    - **values:** Son ficheros XML donde se recogen las cadenas de texto de la interfaz, los colores usados, o las dimensiones entre otros. Así es posible cambiar un estilo en todas las actividades sin hacerlo de manera manual una a una o se puede transcribir la app a otro idioma incluyendo otro fichero que contenga las cadenas traducidas.

    - **values-11, values14, values-w820dp:** El concepto es igual al de las carpetas *drawable*. Aquí se introducirían valores concretos para los casos en los que se ejecute la aplicación en las versiones 11 o 14 de la API de Android. Estas carpetas no las hemos usado.

- **AndroidManifest.xml:** Es el descriptor de la aplicación que mencionamos antes. Es como un resumen donde se declaran los diferentes permisos que se le asignarán a la app, sus actividades junto con su jerarquía de navegación y propósitos (*intents*), sus servicios asociados y los proveedores de contenido.

- **ic-launcher_web.png:** Imagen que representará a la aplicación en la tienda de apps de Google (Google Play).

- **proguard-project.txt**: Configuración para el ProGuard generada automáticamente por el SDK de Android. ProGuard se usa con el propósito de ocultar y proteger el código fuente de las aplicaciones y, en la medida de lo posible, intentar repeler técnicas de decompilación.

- **project.properties:** Archivo generado automáticamente por el compilador para verificar la versión de la API usada cuando la app se instala en un dispositivo.

❋ Interfaz gráfica

La interfaz gráfica se construye con el método de las Actividades explicado en la sección anterior: Se declaran en el XML los elementos que componen la vista y le agregamos funcionalidad desde las clases Java. Para esta nueva versión, se ha abogado esta vez por una apariencia simple y clara que intente mostrar el mínimo número de elementos en cada actividad (especialmente en la actividad dedicada a mostrar el mapa) esto último con objeto de mantener la limpieza en ellas. También se ha procurado reducir al máximo el número de actividades a mostrar para llevar a cabo cualquier operación. Con esto pretendemos aligerar la tarea de mostrar cualquier recurso y evitar que la navegación entre las diferentes secciones de la aplicación se convierta en algo complicado.

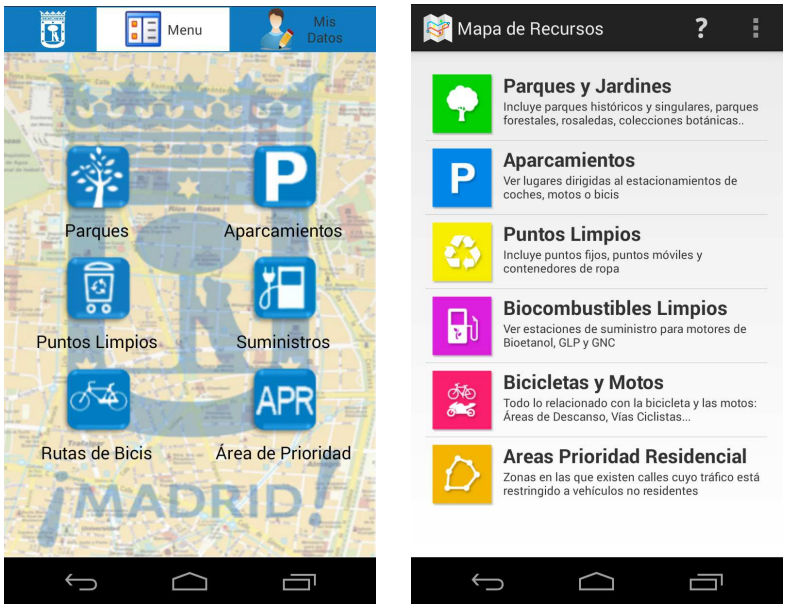


Figura 4.9 – Comparación de la interfaz de la versión 1 (izqda.) con la de la versión 2 (dcha.)

Otra mejora introducida que tiene que ver con la optimización de la navegación entre vistas es la inclusión del elemento ActionBar, que no es más que la barra que aparece siempre en la parte superior de todas las actividades. Su función principal es la de situar al usuario en el contexto de la app y brindarle un conjunto de acciones rápidas relacionadas con el contenido de la actividad.

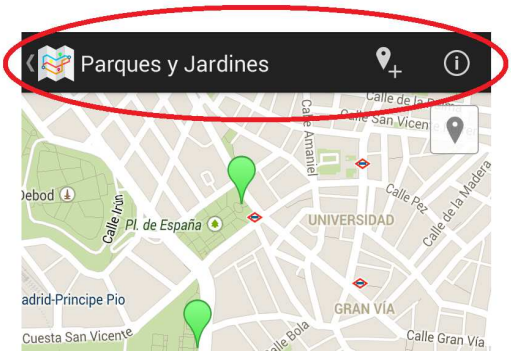


Figura 4.10 – ActionBar *en Android*

Todas las actividades están pensadas para ser mostradas en posición vertical (como ya ocurría en Mapa de Recursos V1), por lo que girar el dispositivo no hará que las actividades roten. En cuanto a los colores, dimensiones, estilos y *strings* usados en estas, se pueden encontrar todos ellos en la carpeta *values* como se ha explicado al principio de esta sección.

Las dimensiones de los objetos de las actividades se proporcionan en dp (*Density Point*), que es la solución que da Google para la fragmentación [61] del tamaño de las pantallas. Dada la diversidad de dispositivos del mercado y sus distintos tamaños de pantalla, no podemos usar píxeles (px) convencionales para los tamaños de los objetos, pues lo que estaríamos diseñando se mostraría mal en cualquier dispositivo que no tuviera ni el tamaño ni la densidad de pantalla que el usado durante las pruebas. Por ello, es necesario establecer unas unidades que hagan estas medidas relativas a esos tamaños y densidades diferentes. De igual manera, hay que repetir el proceso con los tamaños de los textos: Debemos usar sp (*scale-independent points*) en lugar de pt (*Points*), si queremos que estos también se adapten expandiéndose o reduciéndose según se precise. Así es como se consigue el efecto deseado, que las vistas se muestren en todos los dispositivos de la misma manera que se diseñaron y pensaron.

Otra modificación introducida es que el usuario siempre elige cómo mostrar los recursos más cercanos: Si usar su ubicación actual o hacerlo basado en una dirección proporcionada. Con esto evitamos que la ubicación actual sea la opción por defecto como

ocurría en la V1. Esto se ha hecho pensando en los usuarios que no desean activar o no disponen de servicios de ubicación en sus terminales.

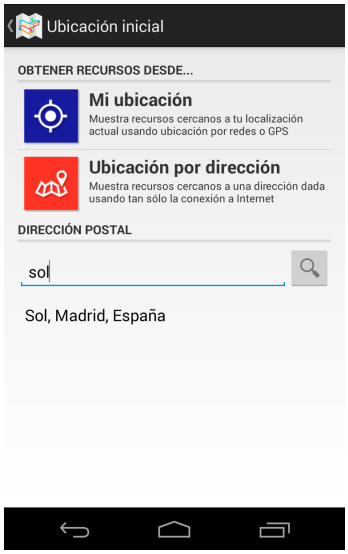


Figura 4.11 – Nuevo menú para la elección de la ubicación base

Finalmente destacar el diseño de los menús auto-descriptivos y la vista de mapa mejorada. Sobre los menús principales, estos se encuentran implementados como objetos *ListView*, es decir, una vista de lista en la que se incluyen las opciones. Decimos que es autodescriptivo dado que cada acción posee un icono ilustrativo, un título que la identifica y un subtítulo con una breve explicación.

Figura 4.12 – Detalle menús descriptivos

En cuanto a las mejoras de la vista de mapa, se ha ampliado el marco que encierra el mapa permitiéndole abarcar toda la pantalla excepto el espacio ocupado por la *ActionBar*. Esta última nos servirá para introducir un botón que permitirá mostrar 10 recursos más cada vez que sea pulsado en lugar de cargar otro nuevo mapa.

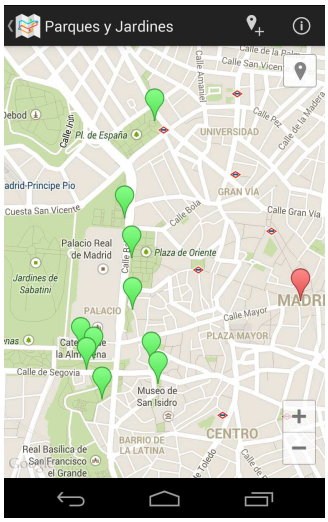


Figura 4.13 – Nueva vista de mapa

❋ Código fuente: Carpetas src y res

En este apartado daremos unas nociones básicas acerca de cómo está el código fuente organizado. Este se encuentra dividido entre las carpetas **src** y **res**. Principalmente el código se encuentra en la primera, y la segunda contiene los *layouts* (capas) que representan las diferentes actividades de la aplicación. Por lo tanto, nos centraremos en el contenido de la carpeta **src** y si se quiere ver qué elementos componen una actividad se debe consultar su fichero XML correspondiente en **res/layout.**

El paquete raíz de la app es: **com.maparecursos.od.** El subpaquete ***od*** son las siglas de Open Data. Dicho subpaquete se encuentra a su vez organizado en otros cuatro paquetes:

- **adapters:** Aquí situamos los adaptadores usados en los objetos del estilo *ListView, GridView,* etc,. durante todas las actividades.

- **app:** En este paquete encontraremos las actividades que representan las diferentes vistas de la misma.

- **app.filters:** Es un subpaquete de app, y vendría a contener más actividades, pero más específicamente, actividades que representan filtros de recursos para el usuario.

- **types:** Contiene los tipos nuevos creados para la aplicación. Aquí estarían, entre otros, objetos que modelan recursos (un parque, un aparcamiento, etc)

Realizaremos un breve repaso de las clases contenidas en estos cuatros paquetes:

➲ Paquete **adapters:**

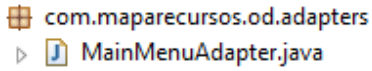


Figura 4.14 – Contenidos del paquete adapters

En nuestro caso sólo incluimos un adaptador, el que se usa para dar forma *(inflar* en jerga Android) a los menús de opciones. Con más detalle, cuando se utiliza un objeto *ListView, GridView*, etc., en alguna actividad, es necesario crear un adaptador para rellenarlo (la lista, la cuadrícula, etc.) con contenido. Los adaptadores pueden ser desde relleno con solo texto, o ir un poco más lejos y definir un *layout* (una capa) para describir dicho adaptador.

Es el caso de nuestra clase **MainMenuAdapter.java**, que rellena los elementos de la lista con un *layout* que modela un objeto con un icono ilustrativo, un título que identifica la opción y un subtítulo con una breve explicación.

➲ Paquete **app:**

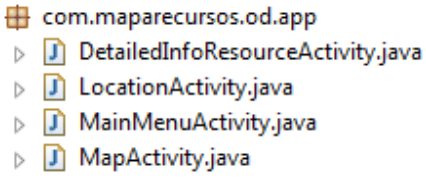


Figura 4.15 – Contenidos del paquete app

El comportamiento de la aplicación se recoge principalmente en cuatro clases:

- **MainMenuActivity.java:** Primera actividad que se muestra al arrancar la aplicación. En esta actividad aparece un menú con las seis categorías de recursos incluidas en la app Mapa de Recursos V2.

- **LocationActivity.java:** Es el menú que se muestra antes del mapa y es donde el usuario decide si desea usar su propia ubicación o proporcionar una dirección.

- **MapActivity.java:** Es la clase que contiene el mapa y muestra los recursos solicitados. Es la clase donde más código se reutilizó. El proceso sigue realizándose como en su antecesora: se piden los recursos solicitados al servidor, se reciben y procesan y a continuación se muestran los 10 más cercanos a la ubicación elegida. El usuario en cualquier momento puede mostrar otros 10 mediante un botón en la *ActionBar*. El ordenamiento de los recursos según la distancia (calculada con la fórmula de *Haversine*) se sigue haciendo con *Quicksort*.

- **DBConnection:** Esta es una subclase privada de la actividad Mapa que es con la que se realiza la conexión al sistema Open Data. El contenido de esta clase lo trataremos explícitamente en la sección **Protocolo de Comunicación.**

- **DetailedInfoResourceActivity.java:** Muestra en una nueva actividad, toda la información detallada de un recurso concreto pulsado sobre el mapa.

➲ Paquete **app.filters:**

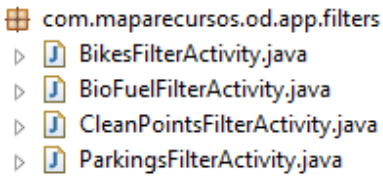


Figura 4.16 – Contenidos del paquete app.filters

Las clases de este paquete son idénticas a **MainMenuActivity.java** salvo que estas son menús que representan filtros para los recursos:

- **BikesFilterActivity.java:** Muestra las diferentes opciones disponibles en la categoría *Bicicletas y Motos***.**

- **BioFuelFilterActivity.java:** Menú para escoger entre los tres tipos de estaciones de carburantes limpios: *Bioetanol, GLP* y *GNC*.

- **CleanPointsFilterActivity.java:** Permite al usuario elegir entre *Puntos Limpios Fijos, Móviles* o *Contenedores de Ropa*.

- **ParkingsFilterActivity.java:** Dividido en tipos de vehículo (*Coche, Moto* o *Bicicleta*) muestra las diferentes opciones de estacionamiento público en la ciudad.

➲ Paquete **types:**

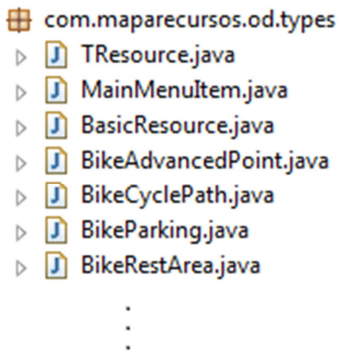


Figura 4.17 – Contenidos del paquete types

Las siguientes clases declaran nuevos tipos de objeto que se incluyen en la aplicación para su funcionamiento. Entre ellos podemos destacar:

- **TResourcejava:** Es un tipo enumerado con los diferentes recursos disponibles para mostrar en la app (Parques, Aparcamientos Coches, Aparcamientos Motos, etc.)

- **MainMenuItem.java:** Clase-registro que encapsula los tres valores que componen una opción en un menú (el icono, el título y el subtítulo).

- **BasicResource.java:** Objeto que encapsula los atributos mínimos de un recurso. Estos serían las coordenadas (latitud y longitud) y la distancia del mismo a la ubicación elegida por el usuario. Además, contiene también la posibilidad de contener listas de coordenadas para algunos recursos especiales que lo requieren, como las vías ciclistas.

- De **BikeAdvancedPoint.java** y en adelante: Son clases que heredan de **`BasicResource`** y que añaden más atributos a los recursos, proporcionando más información como puede ser el nombre, la dirección, etc. Es en estos objetos donde almacenamos la información recibida desde el Open Data.

❋ Android Manifest.xml

Tal como dijimos al comienzo, este fichero es el descriptor del proyecto donde se incluyen las especificaciones de la aplicación. Toda aplicación que se desarrolle en Android debe llevarlo. Cuando se usan entornos integrados como Eclipse, este se encargará de generarlo de manera automática. Realizaremos un breve repaso por las diferentes secciones destacando lo que nos parece más importante:

El elemento padre de este fichero XML es la etiqueta <*manifest*> que contiene a todos los demás elementos hijos incluidos en él. Los atributos a destacar del *manifest* son:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.maparecursos.od"
    android:versionCode="2"
    android:versionName="2.0" >
```

Figura 4.18 – Primera etiqueta de un archivo AndroidManifest.xml

- **android:versionName** y **versionCode:** Hacen referencia al nombre y la versión actual de la app respectivamente. En nuestro caso Mapa de Recursos y versión 2.0
- **package:** Es el paquete principal de la app. En nuestro caso *com.maparecursos.od* donde "od" significa Open Data

El siguiente elemento hijo es <*uses-sdk*>:

```
<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />
```

Figura 4.19 – Elementos que definen versiones mínima y máxima del compilador

Sus atributos definen la versión mínima requerida (API 14: Android 4.0) y la versión del compilador que se usara construir la aplicación (API 19: Android 4.4.2).

El tercer elemento es en el que se declaran los permisos. Para que la app funcione como se ideó es necesario que solicite:

- Acceso a Internet
- Acceso a servicios de ubicación basados en redes
- Acceso a servicios de ubicación basados en GPS
- Acceso de escritura a la unidad externa (normalmente una tarjeta SD o una carpeta en la memoria interna del dispositivo llamada "*sdcard*") para almacenar datos
- Accesos a proveedores de contenidos (en este caso de Google)
- Usar OpenGL 2.0 (no es un permiso, sino activar una característica)

Los tres últimos permisos son necesarios para la inclusión de los mapas de Google. La siguiente imagen recoge los permisos descritos:

```
<!-- Permisos para acceder a Internet -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- Permisos para acceder a la localización por 3G o Wi-Fi -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<!-- Permisos para acceder a la localización por GPS -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<!-- Permisos adicionales para Google Maps (Escritura en unidad externa y Google play
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERV

<!-- OpenGL v.2.0 para GMaps -->
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Figura 4.20 – Elementos que solicitan permisos para la app

Si seguimos, nos encontramos con un par de hijos con etiqueta meta-data que sirven para declarar la *API Key* generada para el uso de los mapas de Google y otra para utilizar los

servicios de Google Play (necesarios desde que Google Maps está asociada a dichos servicios).

```
<!-- Necesario para usar GoogleMaps (Configurar Google Play Services) -->
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />

<!-- GMaps Debug API Key -->
<!--
     DEBUG KEY: AIzaSyAvy0qYwEh_Y58LHj6_x8AjVP-R4X0uqzs
     RELEASE KEY: AIzaSyA33x0BPr5xGge4Sb1dgBaC8c3WfNzxSzs
-->
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyAvy0qYwEh_Y58LHj6_x8AjVP-R4X0uqzs" />
```

Figura 4.21 – Elementos necesarios para la importación de Google Maps

Por último, nos encontramos una sucesión de elementos *activity* (uno por cada actividad de la aplicación) que recogen sus atributos básicos.

```
<activity
    android:name="com.maparecursos.od.app.MainMenuActivity"
    android:label="@string/app_name"
    android:launchMode="singleTop"
    android:screenOrientation="portrait" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
    .
    .
    .
<activity
    android:name="com.maparecursos.od.app.MapActivity"
    android:label="@string/title_activity_map"
    android:launchMode="singleTop"
    android:parentActivityName="com.maparecursos.od.app.LocationActivity"
    android:screenOrientation="portrait" >
</activity>
```

Figura 4.22 – Elementos activity *que componen la app*

Cada elemento <*activity*> debe disponer, al menos, de los siguientes atributos:

- **android:name:** Indica el nombre de la clase dentro del paquete en el código fuente

- **android:label:** Es la etiqueta de texto que se mostrará en la *ActionBar* para identificar la actividad actual al usuario.

Además, se añaden estos atributos opcionales por cuestiones de funcionamiento:

- **android:parentActivityName:** Contiene el nombre de la actividad a la que debe volverse cuando se pulsa el botón *Up* (subir) de la ActionBar.

- **android:launchMode:** Describe como deben iniciarse las actividades. Por defecto, cada vez que se lanza una actividad, Android crea una nueva instancia de la misma. Esto resulta un problema cuando un usuario está visualizando el mapa y pulsa sobre un recurso concreto para ampliar detalles. Para mostrar dichos detalles es necesario lanzar otra actividad como contenedor de visualización. En el momento en el que el usuario decida volver atrás para seguir mirando recursos sobre el mapa, el sistema volverá a lanzar una nueva instancia de la actividad "Mapa", con su correspondiente conexión al Open Data y recálculo de las distancias.

  Nuestra intención en ese caso no es repetir todo el proceso de nuevo, sino volver al estado anterior, y eso se consigue al introducir el valor **singleTop**. Este producirá que las actividades se inicien y se apilen en una pila del sistema, de manera que cuando el usuario quiera volver atrás, se recupera el estado anterior de la actividad.

- **android:screenOrientation:** Sirve para fijar la orientación de la pantalla por defecto. Por comodidad en el diseño, decidimos mantener todas las actividades en posición vertical (*portrait*) como se hacía en la V1.0 de la aplicación.

Para finalizar, hacer mención al grupo que aparece identificado como *intent*. El propósito de dicho código es el de indicar que la actividad *MainMenuActivity* es la primera actividad que se muestra cuando se inicia la app desde el menú del terminal.

❋ Google Maps

Para poder incluir mapas de Google Maps en nuestra aplicación es necesario seguir una serie de pasos en los que aquí no entraremos en detalle, ya que se pueden encontrar de manera actualizada en su web de desarrolladores [62] en el caso de estar interesado en profundizar

en este tema. Sin embargo, si queremos resaltar algunos puntos del proceso por su importancia.

Google quiere controlar los abusos en el uso de su sistema de mapas, por ello exige la obtención de una API Key que identifica cada app que desea acceder al servicio de cartografía. Con esta identificación, Google puede realizar estadísticas, entre las que se encuentra, el seguimiento del número de consultas que se realizan para limitarlo en caso necesario (25.000 / día en la cuenta gratuita).

Por lo tanto, para la obtención de una API Key es necesario disponer de una cuenta en Google. Como disponíamos de la dirección *opendatamadriducm@gmail.com* asociada a este proyecto, obtuvimos una API key con dicha cuenta para la fase de desarrollo. En el momento que se decida lanzar esta aplicación en el *market* de Google, sería recomendable obtener una API Key desde la misma cuenta que administrará dicho lanzamiento, con objeto de centralizar los permisos. Por supuesto, será necesario actualizar el código de la aplicación con la nueva API Key y exportar un nuevo archivo *.apk*.

Otro punto a resaltar es que si siguen fielmente los pasos para introducir los mapas en la app según el tutorial de Google, acabaremos incluyendo la librería de soporte v4, que en nuestro caso, no es necesario. Esto se debe a que la API de mapas de Google está pensada para funcionar con todas las versiones de Android activas y, en las más antiguas, la característica clave para que funcionen los mapas no está incluida.

La característica en cuestión son los *Fragments* [63] (introducidos en la versión 3.0 de Android), los cuales son marcos que sirven para incrustar actividades dentro de otras actividades (de ahí el nombre de *fragmento*). Para el caso concreto de los mapas, Google ha diseñado uno especial denominado ***MapFragment***, que es el marco que contiene el mapa que se verá finalmente en la interfaz. Por lo tanto, en las versiones que no dispongan de acceso a *Fragments (*anteriores a la 3.0 - API 11) deberán incluir la librería de soporte.

Para finalizar, mencionar que las distintas funcionalidades sobre los mapas implementadas (incluida la localización del usuario) en Mapa de Recursos V2 están basadas en los ejemplos oficiales de Google; esto es, en la página de desarrolladores de Google Maps existe un proyecto Android, con muchos y variados ejemplos de las distintas posibilidades que nos ofrece su API para mapas, cuyo código fuente está disponible para descargar. Gracias a esto se pudieron estudiar e incluso ejecutar los ejemplos para comprender su funcionamiento, y sacar el máximo partido a la API que proporciona Google sobre su servicio de mapas, consiguiendo una aplicación móvil final muy profesional, como si de la misma aplicación de Google Maps se tratase.

4.3 Datasets de las aplicaciones de Medio Ambiente y Movilidad

Esta sección está dedicada a repasar los diferentes conjuntos de datos (*datasets*, en inglés) que servirán de fuente a las variadas aplicaciones del grupo G-Tec, que como ya sabemos, están enfocadas al área de Medio Ambiente y Movilidad. Sin embargo, como hemos venido haciendo desde el principio de este capítulo, nos ocuparemos únicamente de los datasets necesarios para que el funcionamiento de la app de Mapa de Recursos 2.0. Estos datasets forman un amplio abanico de formatos y estilos que deberán ser debidamente tratados antes de su publicación en el sistema Open Data. Dejaremos como tarea pendiente para el resto de aplicaciones, el análisis de los datasets que se precisan para la completa integración de cada una de estas aplicaciones en el sistema Open Data.

4.3.1 Datasets necesarios en Mapa de Recursos Ambientales 2.0

Para nuestro caso de estudio, como ya mencionamos durante el proceso de integración, fue necesario asistir a reuniones con personal del Ayuntamiento en las que solicitamos copias actualizadas de toda la información necesaria. Al principio, nos proporcionaron los datos pedidos por correo electrónico. Normalmente llegaban en ficheros CSV o XLS (Hoja de cálculo de Microsoft Excel) y en raras ocasiones en formato *ShapeFile* (SHP). Sin embargo, puesto en marcha el Portal de Datos Abiertos oficial del Ayuntamiento Madrid en abril de 2014, comenzamos a usar los ficheros disponibles en dicho portal. Muchos de estos ficheros eran prácticamente los mismos que los que ya obraban en nuestro poder, aunque, disponibles en más formatos que analizaremos con detalle en la siguiente sección.

Para conocer los datasets necesarios debemos remitirnos al funcionamiento de la app. Recordando brevemente, el objetivo de la misma es el de localizar los recursos y servicios municipales —concretamente, los más cercanos en base a una ubicación proporcionada por el usuario— disponibles para los ciudadanos en la ciudad de Madrid. Dichos recursos se encuentran repartidos en seis categorías:

- Parques y Jardines
- Aparcamientos
- Puntos Limpios
- Estaciones de Suministros Limpios
- Bicicletas y Motos

- Áreas de Prioridad Residencial (APR)

Vamos a analizar dichas categorías para ver en detalle los conjuntos de datos que la componen, pero no sin antes resaltar, que independiente de los formatos de los datasets, hablaremos de estos como si fuesen tablas genéricas (siempre y cuando los datos lo permitan), ya que resulta más sencillo para la comprensión humana.

❋ Parques y Jardines

Formatos disponibles: CSV, RDF, GEO y XML

Este es el conjunto de datos más sencillo puesto que no contiene subcategorías. Se compondría de una única tabla con las siguientes columnas. Las filas sombreadas se corresponden con datos que consideramos irrelevantes para utilizar en la app.

COLUMNA	OBSERVACIONES
PK	*Primary Key* (No se usa)
NOMBRE	Nombre del parque o jardín
DESCRIPCION-ENTIDAD	Contiene un breve texto explicativo acerca de las características del parque o jardín en cuestión
HORARIO	En las filas vacías se asume 24 horas. En el resto se detalla el horario al público actual del parque o jardín
EQUIPAMIENTO	Indica las instalaciones de las que dispone el recurso (Zonas infantiles, caninas, etc.)
TRANSPORTE	Contiene indicaciones para llegar en Bus o Metro siempre que sea posible
DESCRIPCION	Da datos más técnicos acerca de quién administra el parque y la vegetación que lo compone
ACCESIBILIDAD	Es un campo que aparece siempre a cero
CONTENT-URL	URL donde visualizar esta información en la página web del Ayuntamiento de Madrid
NOMBRE-VIA	Nombre de la calle, plaza, etc. donde se ubica
CLASE-VIA	Si es calle, plaza, etc.
TIPO-NUM	V: Válido o S/N (Sin número)
NUM	Número en el que está situado dentro de la calle, avda., etc.

PLANTA	Suponiendo un edificio, la planta concreta
PUERTA	Indica si alguna puerta es la concreta para acceder
ESCALERAS	Suponiendo un edificio, escaleras izquierda o derecha, etc. No se usa en este dataset
ORIENTACION	Detalles adicionales de ubicación
LOCALIDAD	En nuestro caso siempre es MADRID
PROVINCIA	En nuestro caso siempre es MADRID
CÓDIGO-POSTAL	Indica el código postal asociado a la dirección
BARRIO	Complementa la dirección indicando el barrio al que pertenece
DISTRITO	Complementa la dirección indicando el distrito al que pertenece
CORDENADA-X	Coordenadas UTM
CORDENADA-Y	
LATITUD	Coordenadas geográficas
LONGITUD	
TELEFONO	Datos de contacto. Muy pocos recursos poseen valores en este campo
FAX	
E-MAIL	
TIPO	Tipo de recurso dentro de la jerarquía de datos del Ayuntamiento

❋ Aparcamientos

En esta categoría hablaremos solo de los estacionamientos dirigidos a coches aunque en la app también incluyamos los de motos y bicicletas. Los correspondientes a estos dos últimos los abordaremos en su apartado Bicicletas y Motos. Cabe mencionar que los aparcamientos de coches suponen una novedad introducida en Mapa de Recursos 2.0, dado que en su antecesora se preparó la sección aunque nunca estuvo operativa; los datos no llegaron a tiempo para poder ser incluidos.

➲ Aparcamientos para Coches

Formatos disponibles: CSV, RDF, GEO y XML

Este dataset se compone de dos tablas:

- **Aparcamientos Públicos + Mixtos:** Que sería una relación de parkings de titularidad pública junto con estacionamientos que también lo son, pero en los que existen un número de plazas reservadas a residentes del barrio o distrito.
- **Aparcamientos P.A.R + Mixtos:** Aquí se encontrarían todos los parkings dedicados exclusivamente a residentes junto con los mixtos descritos antes.

Ambas tablas tiene una composición idéntica:

COLUMNA	OBSERVACIONES
PK	*Primary Key* (No se usa)
NOMBRE	Indica el nombre del estacionamiento y además si es público, mixto o de residentes.
DESCRIPCION-ENTIDAD	No se usa
HORARIO	Solo está especificado en algunos casos
EQUIPAMIENTO	No se usa
TRANSPORTE	Solo está especificado en algunos casos
DESCRIPCION	Contiene datos acerca del número de plazas, los horarios, la titularidad, etc.
ACCESIBILIDAD	Es un campo que aparece siempre a cero
CONTENT-URL	URL donde visualizar esta información en la página web del Ayuntamiento de Madrid
NOMBRE-VIA	Nombre de la calle, plaza, etc., donde se ubica
CLASE-VIA	Si es calle, plaza, etc.
TIPO-NUM	V: Válido o S/N (Sin número)
NUM	Número en el que está situado dentro de la calle, avda., etc.
PLANTA	Suponiendo un edificio, la planta concreta
PUERTA	Indica si alguna puerta es la concreta para acceder

ESCALERAS	Suponiendo un edificio, escaleras izquierda o derecha, etc.
ORIENTACION	Detalles adicionales a toda la dirección compuesta por los campos anteriores
LOCALIDAD	En nuestro caso siempre es MADRID
PROVINCIA	En nuestro caso siempre es MADRID
CÓDIGO-POSTAL	Indica el código postal asociado a la dirección
BARRIO	Complementa la dirección indicando el barrio al que pertenece
DISTRITO	Complementa la dirección indicando el distrito al que pertenece
COORDENADA-X	Coordenadas UTM
COORDENADA-Y	
LATITUD	Coordenadas geográficas
LONGITUD	
TELEFONO	Datos de contacto. Muy pocos recursos poseen valores en este campo
FAX	
E-MAIL	
TIPO	Tipo de recurso dentro de la jerarquía de datos del Ayuntamiento

❋ Puntos Limpios

Aquí encontramos dos tablas, dado que en esta categoría agrupamos los Puntos Limpios junto con los Contenedores de Ropa.

➲ Puntos Limpios Fijos y Móviles

Formatos disponibles: CSV, RDF, GEO y XML

La información de los Puntos Limpios se encuentra compactada en una única tabla. Corresponden a los 16 puntos limpios fijos y a los 16 puntos móviles (uno de cada distrito) disponibles en la ciudad.

COLUMNA	OBSERVACIONES
PK	*Primary Key* (No se usa)
NOMBRE	Indica el nombre del punto limpio así como si es fijo o móvil
DESCRIPCION-ENTIDAD	No se usa
HORARIO	Solo especificado para los P.L. Fijos
EQUIPAMIENTO	No se usa
TRANSPORTE	Solo especificado para los P.L. Fijos
DESCRIPCION	En el caso de los P.L. Fijos informa acerca de cómo deber hacerse el traslado de los residuos y para los P.L. Móviles su localización y horario
ACCESIBILIDAD	Es un campo que aparece siempre a cero
CONTENT-URL	URL donde visualizar esta información en la página web del Ayuntamiento de Madrid
NOMBRE-VIA	Nombre de la calle, plaza, etc. donde se ubica (solo P.L. Fijos)
CLASE-VIA	Si es calle, plaza, etc. (solo P.L. Fijos)
TIPO-NUM	V: Válido o S/N (Sin número)
NUM	Número en el que está situado dentro de la calle, avda, etc (solo P.L. Fijos)
PLANTA	Suponiendo un edificio, la planta concreta
PUERTA	Indica si alguna puerta es la concreta para acceder (solo P.L. Fijos)
ESCALERAS	Suponiendo un edificio, escaleras izquierda o derecha, etc. (solo P.L. Fijos)
ORIENTACION	Detalles adicionales de ubicación (solo P.L. Fijos)
LOCALIDAD	En nuestro caso siempre es MADRID
PROVINCIA	En nuestro caso siempre es MADRID
CÓDIGO-POSTAL	Indica el código postal asociado a la dirección (solo P.L. Fijos)

BARRIO	Complementa la dirección indicando el barrio al que pertenece (solo P.L. Fijos)
DISTRITO	Complementa la dirección indicando el distrito al que pertenece (solo P.L. Fijos)
COORDENADA-X	Coordenadas UTM (solo P.L. Fijos)
COORDENADA-Y	
LATITUD	Coordenadas geográficas (solo P.L. Fijos)
LONGITUD	
TELEFONO	Datos de contacto. No se usan
FAX	
E-MAIL	
TIPO	Tipo de recurso dentro de la jerarquía de datos del Ayuntamiento

➲ Contenedores de Ropa

Formatos disponibles: XLS

En cuanto a la tabla de contenedores de ropa, la estructura ya no es igual que las anteriores, sino bastante más simple:

COLUMNA	OBSERVACIONES
DISTRITO	Complementa la dirección indicando el distrito al que pertenece
CENTRO	Indica el nombre del centro (Un colegio, un punto limpio, etc.) al que se encuentra asociado el contenedor de ropa
TIPO-VIAL	Si es calle, plaza, etc.
PARTICULA-VIAL	Palabra auxiliar que une el tipo de vial con el nombre del vial. Ejemplo: Paseo **del** Prado
NOMBRE-VIAL	Nombre de la calle, plaza, etc., donde se ubica
NUMERO-VIAL	Número en el que está situado dentro de la calle, avda., etc.
COORD-X	Coordenadas UTM
COORD-Y	

❋ Estaciones de Suministros Limpios

Formatos disponibles: XLS

En este dataset encontramos la relación de estaciones de servicio que disponen de surtidores de biocombustibles limpios. La tabla que compone este dataset mezcla estaciones de Bioetanol, GLP (Gas Licuado de Petróleo) y GNC (Gas Natural Comprimido)

COLUMNA	OBSERVACIONES
TIPO DE COMBUSTIBLE	Aquí se especifican las 3 opciones disponibles: Bioetanol, GLP o GNC
NOMBRE ESTACION	Indica el nombre de la estación de servicio
DISTRITO	Complementa la dirección indicando el distrito al que pertenece
CLASE DE VÍA	Palabra auxiliar que une el tipo de vial con el nombre del vial. Ejemplo: Paseo **del** Prado
CALLE	Nombre de la calle, plaza, etc. donde se ubica
NÚMERO	Número en el que está situado dentro de la calle, avda., etc.
UTM-X	Coordenadas UTM
UTM-Y	
LATITUD	Coordenadas geográficas
LONGITUD	
COMERCIALIZADOR	Empresa a cargo de la administración de la estación de servicio.

❋ Bicicletas y Motos

Formatos disponibles: ZIP/SHP

Esta es una categoría especial, ya que reúne un conjunto de servicios variados que ofrece la ciudad de Madrid tanto para ciclistas como para motoristas. Cada tipo de vehículo genera los siguientes datasets:

- **Bicicletas**
    - **Aparcabicis**: Zonas destinadas al estacionamiento de bicicletas
    - **Áreas de descanso**: Indican puntos del Anillo Verde Ciclista destinados al descanso de los ciclistas. Estas zonas disponen de espacios para aparcar las bicis, fuentes de agua, etc.
    - **Avanza Bici:** Es una iniciativa para mejorar la visibilidad de las bicicletas en los semáforos por parte del resto de vehículos.
    - **Calles Tranquilas:** Relación de calles que no son necesariamente una vía ciclista, pero que debido a la baja intensidad de tráfico que registran son indicadas para el tráfico de bicicletas.
    - **Vías Ciclistas:** Relación de vías destinadas al uso de bicicletas.
- **Motos**
    - **Avanza Moto:** Iniciativa similar a Avanza Bici, aunque incluye algunos puntos adicionales destinados exclusivamente a motoristas.
    - **Reservas Moto:** Zonas destinadas al estacionamiento de motos y motocicletas.

Uno de los motivos de por qué es especial esta categoría se debe a que esta sección es una ampliación de la que ya había dedicada exclusivamente a bicicletas en Mapa de Recursos 1.0. Entonces sólo contenía Vías Ciclistas y Calles Tranquilas. Los servicios de Áreas de Descanso, Avanza Bici y Avanza Moto son otra de las novedades en Mapa de Recursos 2.0.

La otra razón por la que esta categoría es especial es porque ninguno de estos *datasets* está disponible en ningún tipo de tabla, sino que la información se encontraba en un formato de fichero desconocido hasta el momento, el *ShapeFile*. En la siguiente sección sobre formatos describiremos en profundidad las características de este tipo de ficheros. De momento nos basta con saber que es un formato que sirve para mostrar datos geográficos. Consiste en coger un mapa base, y después añadirle una capa en la que podemos situar puntos, polilíneas o incluso polígonos para delimitar ciertas localizaciones, vías, emplazamientos, etc. Más tarde, en la sección de Normalización descubriremos cómo transformar este tipo de ficheros en una tabla y el porqué de esta necesidad.

❋ Áreas de Prioridad Residencial

Formatos disponibles: PDF

Este es otro caso especial. Las Áreas de Prioridad Residencial (A.P.R) son la solución del Ayuntamiento para reducir la densidad del tráfico en ciertas zonas, así como la contaminación atmosférica y acústica. Otro de los propósitos de estas áreas es el de aumentar las plazas de aparcamiento para los residentes así como ordenar las zonas de carga y descarga en la vía.

A estas áreas pueden acceder sin restricciones los residentes, transportes públicos y servicios de emergencia. Las motocicletas podrán hacerlo de 7 a 22. Los horarios de carga y descarga dependen de la APR.

Concretamente son tres áreas las que existen en la ciudad de Madrid y la información referente a ellas se encuentra en unos archivos en formato PDF correspondientes al proyecto Mapa de Recursos 1.0. Como esta información no había sufrido cambios, se usaron estos mismos ficheros.

4.4 - Formatos de los Datasets

En esta sección realizaremos un recorrido muy somero por los diferentes formatos en los que se encuentran los conjuntos de datos de la aplicación Mapa de Recursos 2.0 y mostraremos algunos ejemplos.

❋ Formato CSV

El formato **CSV** [64] (en inglés: *Comma-Separated Values*; en castellano: Valores separados por comas) es un formato abierto y sencillo que permite representar datos como si se tratase de una tabla. En el estándar CSV original las filas se representan por saltos de línea y las columnas tienen que separarse por una coma (“,”), aunque en países como España, Francia, Italia, etc., se usa como separador el punto y coma (“;”). Si algún valor contiene comas, o saltos de línea, entonces se encerrará dicho valor con comillas dobles (“valor, con comas”).

Un ejemplo sencillo de fichero CSV::

```
NOMBRE,APELLIDOS,ALTURA,EDAD
Marcos,Perez Dominguez,16,”1,65”
Maria,Pulido Arquero,17,”1,72”
```

❋ Formato XLS

Los archivos con extensión XLS son un formato propietario de Microsoft que sirven para representar hojas de cálculo (tablas con filas y columnas) mediante el software Microsoft Excel. La principal característica que diferencia los CSV de los XLS (pues a simple vista, ambos modelan tablas) es que, los XLS permiten la introducción de operaciones aritméticas en las celdas, cuyos operandos son a su vez, otras celdas. Aquí es donde entra en juego el programa Excel, ya que hace de intérprete de dichas operaciones, y automáticamente muestra en las celdas los resultados de dichas fórmulas. Las posibilidades del software van mucho más allá, dado que permite generar también gráficos a partir de los valores u operaciones de las celdas. De esta manera, Excel proporciona una herramienta sencilla para crear un libro contable, una plantilla para facturas, inventarios, etc.

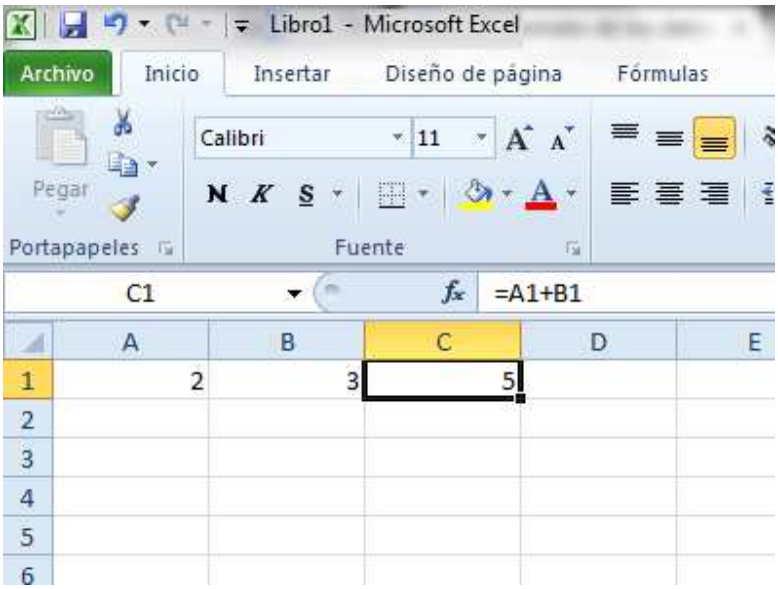


Figura 4.23 – Ejemplo de formula simple en Microsoft Excel

❋ Formato XML

XML [65] (acrónimo en inglés: *eXtensible Markup Language*, en castellano: Lenguaje de Marcas Extensible) es otro formato de ficheros de carácter abierto, de los que se denomina semi-estructurado, por su representación a medio camino entre la legibilidad humana y la de la máquina. Es un formato que se creó inicialmente enfocado a Internet, con el objetivo de ampliar la compatibilidad entre diferentes sistemas, pero hoy en día es fácil encontrarlo en bases de datos, servicios web, ficheros de texto, ficheros de configuración, hojas de cálculo, etc.

La estructura típica de un XML es la de un archivo que se va componiendo de etiquetas que encierran valores. Cada etiqueta posee un identificador (el que se desee, he aquí la extensibilidad) y, opcionalmente, atributos asociados a ella. Cada una de estas etiquetas es lo que se denomina **elemento**. Un elemento puede contener subelementos, que se denominan **hijos**; o lo que es lo mismo, etiquetas dentro de etiquetas manteniendo una jerarquía que es de vital importancia para la interpretación del fichero. Un ejemplo sencillo de fichero XML podría ser el siguiente.

```
<Mensaje
      <Remitente atributo pais = ESPAÑA >
Pedro
</Remitente>
<Destinatario atributo pais = ITALIA >
Andrea
      </Destinatario>
      <Contenido>
            ¡Ciao! ¿Que tal va todo?
      </Contenido>
</Mensaje>
```

Una idea de la jerarquía de etiquetas nos la da la tabulación de las mismas, aunque los *parseadores* de XML se basan en las aperturas (<etiqueta>) y cierres (</etiqueta>) de etiquetas para construirla.

❋ Formato KML

KML [66] (siglas de *Keyhole Markup Language*) es un formato de archivo basado en XML especializado en la representación de datos geográficos en tres dimensiones (latitud, longitud y, opcionalmente, altitud). Su origen se debe al precursor (un programa llamado Keyhole LT) de lo que hoy conocemos como Google Earth, que es un software que permite viajar alrededor del mundo y visualizar en un mapa diferentes lugares de la tierra a través de fotografías tomadas por satélite o incluso a pie de calle (Google Street View). Con KML se pueden añadir detalles a los mapas, como puntos, líneas, polígonos, imágenes, etc., que ayuden a situar o resaltar lugares en el mapa.

Aunque este formato no se corresponde con ninguno de los datasets explicados en la sección anterior, lo incluimos aquí porque haremos mención a él durante el proceso de normalización. Un ejemplo de KML mostrando un punto que señalice un parque sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" >
      <Placemark>
            <name>Parque Plaza de España </name>
            <description> Parque de la zona centro de Madrid con
                        monumento y lámina de agua. Es un lugar muy
                        visitado por turistas… </description>
      <Point>
            <coordinates>
            40.41641041087390, -3.662606100328210
            </coordinates>
      </Point>
</kml>
```

❋ Formato ShapeFile (SHP)

El formato **ESRI-ShapeFile** [67] es un formato de archivo propietario perteneciente a la compañía ESRI, especializada en software orientado a Sistemas de Información Geográfica (en adelante, abreviado SIG), y que al igual que KML, también sirve para almacenar datos geográficos. El propósito es bastante similar: añadir detalles a los mapas como puntos, líneas, etc para resaltar lugares. La diferencia es que en lugar de usar XML, utiliza un formato vectorial donde almacena los datos de los elementos añadidos al mapa.

El uso de estos archivos está bastante extendido en los SIG y, de hecho, casi se podrían considerar un estándar en el sector. Están pensados para ser tratados con la suite ArcGis, que quizá es el software más utilizado cuando se trabaja con SIG. Esto se debe principalmente a la larga experiencia que tiene el grupo ESRI en el mercado, lo que le ha hecho colocarse en una importante posición.

Como ya adelantamos de manera más simple durante la sección dedicada a los datasets, un ShapeFile no posee la capacidad de almacenar datos topológicos; esto es, los datos se construyen y tienen relación con un mapa base, pero este no está incluido en el fichero SHP. Por eso decíamos que lo podríamos considerar una capa, y esta sólo tiene sentido cuando se aplica sobre el correspondiente mapa base.

En realidad, el formato SHP no es un formato de archivo, sino multi-archivo; en otras palabras, se compone de varios ficheros. De hecho un SHP suele ser una carpeta comprimida (normalmente ZIP) que contiene entre 3 y 11 ficheros. Tres es el mínimo de archivos que componen un SHP y los otros ocho ficheros restantes son para ajustes opcionales. Si descomprimimos cualquiera de los datasets que se encuentran en formato SHP; por ejemplo, el de Áreas de Descanso para Bicicletas, podemos encontrar lo siguiente:

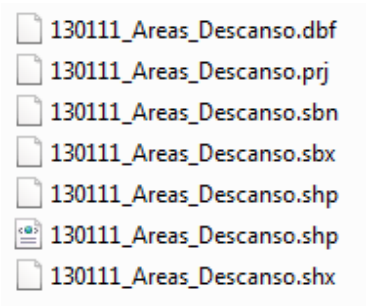


Figura 4.24 – Ejemplo de estructura multiarchivo en un ShapeFile

- Ficheros requeridos
    - **.shp:** Contiene todos los elementos geométricos añadidos al mapa.
    - .**shx:** Contiene los índices de dichos elementos.
    - **.dbf:** Es la base de datos que contiene la información de los objetos de la capa.
- Ficheros opcionales
    - **.prj**: Es el archivo que guarda la información del sistema de coordenadas con formato WKT (un formato para especificar figuras geométricas)
    - **.sbn** y **.sbx** : Contienen los índices espaciales de los objetos representados.
    - **.shp.xml:** Para almacenar metadatos.

Otro tema importante de los SHP es el sistema de coordenadas que usa y al que hacen referencia los objetos representados en él. Por cuestiones obvias, este debe ser el mismo que use el mapa base utilizado, sino los objetos no se podrían situar correctamente en el mapa. Como veremos durante el proceso de normalización esto nos ha supuesto más de un problema en algunos datasets, lo que nos ha llevado a decidir la inclusión de una sección dedicada a tratar los sistemas de coordenadas.

❋ Formato GEO

GEO [68] es un microformato implementado con XHTML (una variante de XML) y que se utiliza para el marcado de coordenadas geográficas en el sistema WGS84. La especificación

existente de dicho microformato se considera provisional, aunque su uso está bastante extendido y aceptado.

La característica principal de GEO consiste en ser un apoyo para los analizadores sintácticos que se dedican a procesar sitios web en busca de coordenadas geográficas. Una vez localizadas, se pueden hacer operaciones con ellas, cargarlas en un sitio web, en un GPS, etc.

Un ejemplo del uso de GEO lo encontramos en el dataset de los aparcamientos para coches, Mostramos un fragmento resumido, pues el archivo contiene una entrada de estas para cada estacionamiento:

```
<entry>
       <title>Aparcamiento público. Almagro</title>
       <link
           href="http://www.madrid.es/vgn-ext-
templating/v/index.jsp?vgnextchannel=9e4c43db40317010VgnVCM100000dc0ca8c0
RCRD&amp;vgnextoid=0e167339ae51c010VgnVCM2000000c205a0aRCRD"
           rel="alternate" type="text/html"/>

       <id>13452</id>
       <content type="html">Plazas : 462  Abierto 24 horas  Admite
Tarjeta Bono Red   Titularidad : Ayuntamiento de Madrid </content>
       <category scheme="$"
term="http://datos.madrid.es/egob/kos/entidadesYo..."/>
       <georss:point>40.43030345785862Â -
3.6931471299523446</georss:point>
       <geo:lat>40.43030345785862</geo:lat>
       <geo:long>-3.6931471299523446</geo:long>
</entry>
```

❋ Formato RDF

Los archivos **RDF** [69] (siglas en inglés de *Resource Description Framework*; en castellano: Marco de Descripción de Recursos) son contenedores de metadatos para la descripción de los recursos de un sitio web. La notación usada para describir dichos recursos suelen ser la de los denominados tripletes con estructura *sujeto-predicado-objeto*, donde el sujeto es el recurso que se describe, el predicado es donde se fijan las propiedades o relaciones y en el objeto se define el valor de dichas propiedades o recursos con los que están relacionados. El uso en conjunto de los RDF con otras herramientas constituye uno de los pilares básicos de lo que se conoce como Web Semántica.

Para ilustrar un RDF, utilizaremos un fragmento reducido del dataset Parques y Jardines en el que se añaden los metadatos de un jardín en concreto:

```
 <v:VCard
rdf:about="http://datos.madrid.es/egob/catalogo/tipo/entidadesyorganismos
/5965967-jardines-basilica">
       <v:fn>Jardines de la Basílica</v:fn>
       <rdf:type
rdf:resource="http://datos.madrid.es/egob/kos/entidadesYorganismos/Parque
sJardines"/>
       <dc:relation>http://www.madrid.es/vgn-ext-
templating/v/index.jsp?vgnextchannel=9e4c43db40317010VgnVCM100000dc0ca8c0
RCRD&amp;vgnextoid=58df433bf683e210VgnVCM1000000b205a0aRCRD</dc:relation>
       <v:org>
           <rdf:Description>
               <v:organisation-name>Jardines de la
Basílica</v:organisation-name>
               <org:horario/>
               <org:edificioaccesible>No</org:edificioaccesible>
               <org:servicios/>
               <dc:description> Situados frente a la Basílica
Hispanoamericana de ... </dc:description>
           </rdf:Description>
       </v:org>
       <v:adr>
           <rdf:Description>
               <loc:distrito
rdf:resource="http://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/
Madrid/Distrito/Tetuan"/>
               <loc:barrio
rdf:resource="http://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/
Madrid/Distrito/Tetuan/Barrio/CuatroCaminos"/>
               <v:street-address>CALLE BASILICA 15</v:street-address>
               <v:locality>MADRID</v:locality>
               <v:postal-code>28020</v:postal-code>
           </rdf:Description>
       </v:adr>
       <v:geo>
           <rdf:Description>
               <geo:long
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
3.695615530710886</geo:long>
               <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">40.44903638127448<
/geo:lat>
           </rdf:Description>
       </v:geo>
   </v:VCard>
```

❋ Formato PDF

PDF [70] (siglas de *Portable Document Format* en inglés; en castellano: Formato de documento portable) es un formato orientado a la creación de documentos digitales que puedan ser leídos en cualquier máquina, independientemente del hardware o software que utilice esta. Que sea multiplataforma es la característica esencial a la que debe su uso tan

extendido en Internet para el intercambio de documentos. Se dice que es una formato compuesto al combinar en un mismo archivo imágenes vectoriales con mapas de bits y texto. Aunque su creador es la empresa privada Adobe Systems, actualmente se considera un formato abierto y estandarizado.

Un ejemplo sencillo de archivo PDF podría ser la edición digital de esta memoria. Al exportarla a dicho formato, permitimos que pueda ser leída en cualquier máquina (multiplataforma), siempre y cuando se disponga de un lector de PDF's en la misma.

4.5 - Sistemas Geodésicos de Referencia

Antes de comenzar de lleno con el proceso de normalización de los datasets, debemos hacer un alto en el camino para tratar un problema en el que se ven implicados tanto los conjuntos de datos que contengan datos geográficos como las aplicaciones que basen sus fuentes en dichos datasets: Es necesario unificar los criterios en cuanto a qué sistemas de coordenadas se usarán en las proyecciones y en los datos, de manera que se obtengan siempre los resultados esperados.

Este problema se vuelve especialmente dramático para la app Mapa de Recursos, ya que basa todo su funcionamiento en mostrar mapas en los que se ubican elementos. Se espera que dichos elementos aparezcan correctamente situados y no se desvíen por una mala concordancia entre el sistema de coordenadas del mapa sobre el que mostramos los recursos y las coordenadas de los elementos del dataset en cuestión.

Con mala concordancia nos referimos a que los sistemas de coordenadas del mapa base y los datasets no sean los mismos. Por ello, como veremos más adelante, serán necesarios cambios y transformaciones entre dichos sistemas para que al final todo funcione perfectamente. A continuación, vamos a hablar de los Sistemas Geodésicos de Referencia en los que se inscribe nuestra aplicación y sus conjuntos de datos. Concretamente:

- Mapa de Recursos utiliza como mapas base los que proporciona la API de Google Maps que siguen el sistema de coordenadas **WGS84**.

- Los conjuntos de datos contienen coordenadas geográficas en multiples sistemas: **WGS84**, **ETRS89** o **ED50**.

Por lo tanto, para la correcta ejecución de la app será necesario transformar todos los datos geográficos incluidos en el dataset al sistema WGS84 que es el formato de los mapas de Google. Hemos decidido dicho marco de referencia puesto que la API de Google Maps es multiplataforma y gratuita, está bastante bien documenta y es fácil de usar, por lo que podría considerarse una buena opción para empezar en cualquier aplicación. También se pueden elegir otras APIs de mapa, siempre y cuando el mapa base también use proyección

WGS84. Comenzaremos haciendo algunas definiciones previas y después analizaremos todos estos sistemas de coordenadas en detalle [71].

❋ Figura de la Tierra

Primero empezaremos definiendo la figura natural de la Tierra, en la que no tendremos en cuenta su relieve, de manera que se asemeje a la forma de un geoide, el cual es una "superficie de nivel equipotencial al campo gravitatorio terrestre". La superficie de referencia para la altitud será el nivel medio de los océanos cuando no se tiene en cuenta la acción perturbadora de las mareas.

En palabras mucho más simples, vendremos a imaginarnos la Tierra como si fuese una simple pelota redonda, aunque no necesariamente esférica, dado que la Tierra tiene forma elipsoidal. Esto es lo que trataremos ahora:

❋ Sistemas Elipsoidales de Referencia

Como nuestro resumen es demasiado sencillo y la definición matemática de geoide es demasiado compleja, debemos alcanzar un punto intermedio que sea un poco más formal pero también fácil de comprender. Una buena aproximación es representar la superficie de nuestra figura de la tierra como un elipsoide de revolución. Así podemos definir el siguiente sistema elipsoidal como:

- Superficie de referencia: dimensiones (semiejes a,b).
- Ejes o líneas de referencia en la superficie
- Sentidos de la medida

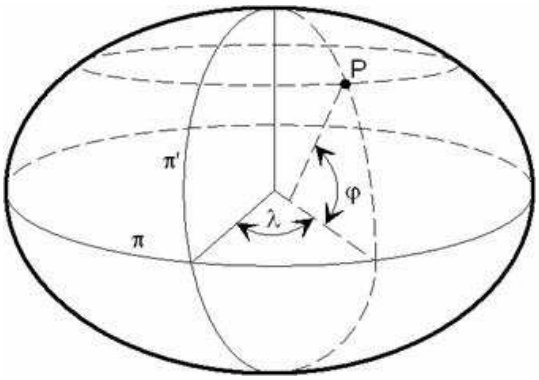


Figura 4.24 – Ejemplo de elipsoide de revolución para representar La Tierra

Una vez hemos definido el sistema, podemos ahora introducir las coordenadas geodésicas:

- **Latitud geográfica** (ϕ): Ángulo, medido sobre el plano meridiano que contiene al punto, entre el plano ecuatorial y la normal al elipsoide en el punto P.

- **Longitud geográfica** (λ): Ángulo, medido sobre el plano ecuatorial, entre el meridiano origen y el plano meridiano que pasa por el punto P.

La elección del elipsoide de revolución concreto como aproximación del geoide depende un poco de la zona del planeta que queramos representar, por ello, a lo largo de la historia, cada país ha ido utilizando aquel que mejor se adaptaba a su área de confluencia. Todos ellos han ido formando lo que se conoce como el conjunto de los Sistemas de Referencia.

En geodesia existirán dos *Datums*: el horizontal y el vertical. Este último vendría a ser la superficie que tomaremos como referencia para las medidas de las altitudes. En tal caso, lo normal es que dicha superficie sea el propio geoide.

Ahora que ya tenemos definido como es un Sistema de Referencia podemos empezar a hablar de algunos concretos, como los mencionados al principio de esta sección.

❋ ED50: European Datum 1950

Es un datum descrito por Hayford en 1924 durante la celebración de la Asamblea Internacional de Geodesia y Geofísica en Madrid. En él propone un Elipsoide Internacional de Referencia con unos primeros valores de los semiejes a y b. Dicho elipsoide fue ampliamente aceptado en la mayoría de países pese a no ser perfecto: En 1964, la Unión Astronómica Internacional (Hamburgo) propuso unos nuevos valores para realizar unos ajustes.

Las normas de orientación para este sistema son:

- El eje menor del elipsoide de referencia es paralelo a la dirección definida por el origen internacional convencional (O.I.C) para el movimiento del polo.

- El meridiano de referencia es paralelo al meridiano cero (Greenwich) adoptado por el BIH para las longitudes.

Este datum es el que se utiliza en España de manera oficial desde 1970, año en que sustituyó al Elipsoide de *Struve* y al Datum Madrid del Observatorio del Retiro. Cabe destacar que pese a ser adoptar el sistema 6 años después de las correcciones, los valores

para dicho sistema son los que se definieron en 1924 por Hayford debido a que se ajustan mejor a la zona de la península.

❋ WGS84: World Geodetic System 1984

El **Sistema Mundial Geodésico 1984** es un datum que cobra popularidad con la llegada en 1987 del GPS, al ser este el sistema al que está referenciado. Se trata de un sistema único para referenciar posiciones terrestres, individualmente o agrupadas en vectores. Está basado en observaciones *Doppler* desde el sistema de satélites de navegación NNSS o *Transit*, intentando que fuese lo más fielmente a la forma de toda la Tierra.

WGS84 se representa con un sistema cartesiano geocéntrico:

- **Origen:** Centro de masas de la Tierra (incluyendo océanos y atmósfera).
- **Eje Z:** Paralelo a la dirección del polo CIO o polo medio definido por el BIH. Los parámetros serían época 1984.0 y precision 0.005"
- **Eje X:** Intersección del meridiano cero (Greenwich) con el plano que pasa por el origen y es perpendicular al eje Z.
- **Eje Y:** Ortogonal a los anteriores, pasando por el origen.
- Terna rectángulo dextrosum

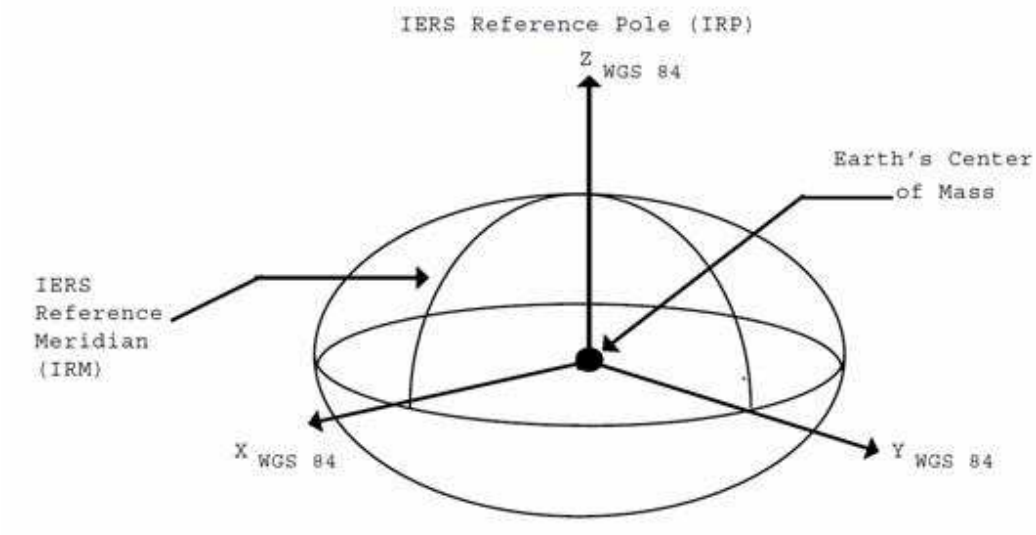


Figura 4.24 – Esquema del elipsoide de revolución definido por el sistema WGS84

❋ Marcos y Sistemas de Referencia Terrestres: El International Terrestrial Reference Frame (ITRF).

Para materializar un marco geodésico mundial de referencia hay que establecer una serie de puntos con sus coordenadas, los cuales deben poder inferir:

- La localización de un origen
- La orientación del sistema de ejes cartesianos ortogonales
- Una escala

Actualmente, los Marcos de Referencia Terrestres (en inglés TRF, *Terrestrial Reference Frame*) se obtienen a partir de un grupo de estaciones distribuidas globalmente con unas coordenadas bien definidas. Como consecuencia de los efectos del tiempo en las placas, se establecen sistemas y marcos en referencia a épocas concretas. Hasta la fecha existen 11 ediciones publicadas del ITRF. En resumen, un ITRF es un conjunto de estaciones con sus coordenadas definidas en una época de referencia con sus respectivas velocidades anuales.

❋ ETRS89: Los sistemas European Terrestrial Reference System 1989 y REGCAN95.

El **ETRS89** es el datum que la Subcomisión de la Asociación Internacional de Geodesia (IAG) sugirió en Florencia (1990) como estándar para el marco de referencia europeo (EUREF). El origen de este datum se encuentra en las campañas IBERIA95 y BALEAR98 calculadas a partir del ITRF96 época 1995,4 y 1998,3 respectivamente.

Desde la entrada en vigor el 27 de julio de 2007 del Real Decreto 1071/2007, se establece ETRS89 como el sistema geodésico oficial de referencia para el Reino de España [72], aunque sólo para el caso de la Península Ibérica y las Islas Baleares. Las Islas Canarias se acogen al datum **REGCAN95** debido a que el rango de alcance de ETRS89 está limitado a la placa euroasiática. La definición de REGCAN95 está basada en el ITRF93 cuyos resultados se calcularon en una estación situada en Maspalomas.

Una vez que hemos definido los tres marcos de referencia implicados en nuestros conjuntos de datos y aplicaciones, pueden surgirnos algunas cuestiones que trataremos en los siguientes apartados [73]:

➲ ¿Por qué cambia el Sistema Geodésico Oficial de España?

El auge en el desarrollo de la tecnología GPS durante los 80, llevaron a la IAG a la promoción de un nuevo sistema de referencia único para ser adoptado por todos los países del continente europeo. Para ayudar a consolidar tal objetivo, a partir de 1999 la Comisión Europea lo materializa de forma oficial incluyéndolo en su Normativa Comunitaria. Por esta razón se introduce en España el Real Decreto 1071/2007, el cual recoge entre otras cosas: *"…Toda la cartografía y bases de datos de información geográfica y cartográfica producida o actualizada por las Administraciones Públicas deberá compilarse y publicarse conforme a lo que se dispone en este Real Decreto a partir del 1 de enero de 2015, etc."*

➲ ¿Existe una transformación exacta entre ED50 y ETRS89?

La respuesta teórica es **Sí**, dado que lo único que diferencia dos datums son los parámetros que configuran el elipsoide, por lo tanto, con unas cuantas expresiones matemáticas que representen una serie de traslaciones, rotaciones y un factor de escala, debería ser posible realizar unos cálculos y extraer la transformación.

Sin embargo, en la práctica **no** existe una relación perfecta y este se debe, entre otros factores, a los métodos de observación seguidos, el equipo técnico y humano implicados en las mediciones, etc. Por esta razón, se pueden observar algunas heterogeneidades cuando se realizan transformaciones entre estos datums, más tratándose de ED50 que basa sus medidas en estaciones terrestres y alguna espacial. En cualquier caso, si se requiere una transformación que respete la precisión original, será necesario hacerlo con una que sea estrictamente conforme.

➲ ¿Por qué ETRS89 y no WGS84?

A efectos prácticos, WGS84 y ETRS89 son equivalentes; esto es, para aplicaciones topográficas o cartográficas sencillas se puede usar uno u otro indistintamente y el resultado será casi el mismo. Si realmente somos estrictos, la afirmación anterior se podría considerar una aberración, pues las diferencias entre ambos datums son bastante importantes.

WGS84 es el sistema estándar de facto del GPS, pero hay que tener en cuenta que ni España, ni Europa poseen ningún tipo de infraestructura que lo materialice al tratarse de un invento de los EE.UU. Esta es la razón por la cual las redes geodésicas europeas especifican sus coordenadas en ETRS89, el cual está basado en marcos internacionales de referencia (ITRF). Se podría decir que es la solución más aproximada al sistema EE.UU pero que nos brinda la independencia del mismo y nos permite disponer de nuestros propios sistemas.

- **¿Un punto tiene una latitud y longitud únicas?**

Sí. Cuando trabajamos con varios datums que difieren en los meridianos de origen y los ecuadores debido al elipsoide usado, un punto real en la Tierra (que siempre será el mismo) se corresponderá con un punto con distintas longitudes y latitudes dependiendo del datum al que se refiera.

- **¿Tengo que actualizar cada año las coordenadas de mis datos en base a las actualizaciones anuales del ITRF?**

No. Precisamente para eso sirve la solución ETRS89, para fijar coordenadas *paradas* en una época (1989) en la que todos nos hemos puesto de acuerdo.

- **¿Porque existen hasta 5 versiones de ED50?**

En realidad se tratan todas del mismo datum pero centradas en una zona concreta, por ejemplo: ED50-Europa_medio o ED50-España. Esto es para lograr que métodos sencillos de transformación de coordenadas (como los de los navegadores GPS domésticos) sean más ajustados, dado que cuanto menos terreno abarque un marco de referencia, más aproximadas serán sus medidas.

Para finalizar, las conclusiones a destacar de esta sección, es que es necesario transformar todos los datasets a un mismo sistema de coordenadas: ETRS89 o, en su defecto, a WGS84. Hemos de mencionar aquí, que dadas las someras equivalencias entre ambos, hace difícil a veces averiguar el sistema de coordenadas real usado en el fichero. Asumiremos para estos casos que están en WGS84, puesto que desde el Ayuntamiento se está procediendo este año 2014 con la migración al nuevo sistema ETRS89 para cumplir con los plazos establecidos (enero 2015) por el Real Decreto de 2007.

4.6 Normalización de los datasets

El problema de la normalización surge cuando deseamos acceder a los datasets mediante la **API REST** de nuestro Sistema Open Data (esto es, sin necesidad de acceder al fichero en bruto) lo que facilita el desarrollo de aplicaciones ajenas al Open Data, pero consumiendo recursos (datos) directamente desde él.

Para materializar esta idea es necesario normalizar los datasets a un formato estructurado. En nuestro caso hemos elegido el formato **CSV** como punto de partida para todos los datasets; esto es, todo dataset que quiera implementar una API para acceder a los datos, debe estar en formato CSV. El hecho de disponer de este conjunto de llamadas en los dataset es lo que nos permite el acceso y manejo estructurado de los datos desde las aplicaciones ajenas al Open Data (en nuestro caso, las apps de Mapa de Recursos).

En las subsecciones siguientes, detallamos qué operaciones han sido llevadas a cabo para normalizar los datasets necesarios para las aplicaciones. Con este objetivo proponemos que el acceso a los datos del Open Data y su reutilización por parte de herramientas de análisis o aplicaciones de terceros no requiera demasiado trabajo adicional, haciendo que sea más sencillo e intuitivo su uso.

En la medida de lo posible, hemos intentado mantener al máximo las estructuras y estados de los archivos para lograr el máximo parecido con los originales. Sin embargo, hemos creído conveniente establecer unas reglas básicas o protocolo mínimo a seguir para la publicación en el gestor de contenidos de los datasets en formato CSV. De esta manera nos aseguramos que al menos todos estos datasets podrán ser tratados indiferentemente de que contengan (al menos en cuanto al formato se refiere). Además, podremos extender el juego de llamadas de la API que provee nuestro gestor de contenidos para que admita consultas SQL empotradas (*SQL Embedded*)

Estas son las normas:

- El juego de caracteres será UTF-8 sin BOM, salvo en excepciones que explicaremos más adelante.
- El carácter separador para las columnas será la coma (,). Los valores se escribirán separados por dicho carácter. En caso de que un valor contenga comas, se encerrará entre comillas superiores dobles: (“va,lor”).
- Los identificadores de columnas se escribirán siempre en mayúsculas. No deben contener nunca espacios ni caracteres especiales. Los únicos caracteres permitidos son alfanuméricos (excepto “**Ñ**” y vocales acentuadas) y el guión bajo como separador.
- Las filas se representarán con simples saltos de línea.
- Si se utiliza algún editor especializado en estos archivos que represente gráficamente las celdas (Microsoft Excel, LibreOffice Calc, etc.), asegurarse que todas son del ***tipo texto sin formato,*** incluido las de contenido numérico.
- Los saltos de línea y los valores de una lista (como las listas de coordenadas) se separarán usando el carácter # dejando un espacio a cada lado (“ # “).
- Si incluyen datos de coordenadas geográficas, estas deberán estar en el sistema ETRS89 o en WGS84 si no es posible lo primero.

Durante los siguientes subapartados comentaremos los pasos que sean seguido para ajustar todos los datasets proporcionados para que se verifiquen las reglas expuestas.

❋ Parques y Jardines

Este dataset está disponible directamente en un CSV, por lo que la información ya está estructurad. Se podría utilizar directamente de no ser por un par de problemas que tiene. Por un lado, la codificación del CSV no es la estándar (delimitado por comas: ",") sino que es un dialecto usado por Microsoft Excel en la versión en castellano (delimitado por puntos y comas ";"). El otro problema está en el formato de los números de las coordenadas: Están con separadores de miles y sin comas para los decimales, lo que hace imposible interpretarlas. El uso de comas para los decimales (notación estándar) en lugar de puntos (notación americana o inglesa) es un problema, ya que es la notación que usa Java para trabajar con los números reales.

El gestor de contenidos de nuestro Open Data solo reconoce ficheros CSV con el delimitador estándar (las comas), por lo que subir directamente este fichero producirá errores. Debemos crear un nuevo CSV con el delimitador correcto y usar para ello un editor de CSV distinto de Microsoft Excel (ya que este no delimita por comas). Analizando distintos editores, descubrimos que no era sencillo alcanzar nuestro objetivo de traducción a un CSV con codificación estándar, hasta que decidimos utilizar el editor de hojas de cálculo de Google Drive, que también trata CSV y lo exporta con la codificación estándar.

Para proceder al ajuste del fichero, creamos una nueva hoja de cálculo en Google Drive e importamos el contenido desde un fichero subido desde el ordenador (el CSV con los Parques y Jardines). Con esto ya aseguramos tener un CSV estándar; nos faltaría tratar el tema del formato de las coordenadas.

Corregir esto último se hace de la siguiente manera: Se asigna a las celdas de las columnas de **LATITUD** y **LONGITUD** el formato **Otros formatos > Formatos de número personalizados....** Dentro del menú emergente, seleccionamos la primera opción que son números sin decimales ni separadores de miles

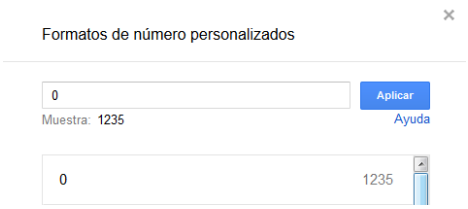


Figura 4.25 – Formato de numero sin decimales en Google Docs

A continuación, volvemos a asignar formato a las celdas, pero esta vez del tipo **Texto sin formato**. El último paso es seleccionar la columna **LATITUD** y usando la **opción Edición > Buscar y Sustituir** junto con expresiones regulares, podemos ajustar las coordenadas automáticamente de la siguiente manera:

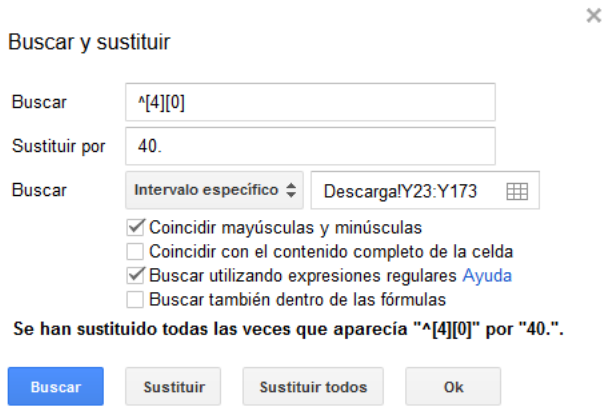


Figura 4.26 – Buscar y Sustituir *utilizando expresiones regulares*

La expresión ^[4][0] es la expresión regular a buscar. Significa "*strings que comiencen por el número 40*". Esto lo hacemos para evitar que si existe un "40" entre los decimales, también le añada un punto. Por ejemplo:

4021354035 => 40.213540.35

Sustituimos por la expresión "**40.**" y pulsamos sobre ***Sustuir todos*** y automáticamente tendremos toda la columna ajustada. Repetimos el proceso para la columna **LONGITUD**, salvo que ahora la expresión regular es ^[-][3] y se sustituye por "**-3.**"

El último detalle a solventar en este fichero se encuentra en la fila 17, columna **TELEFONO**, la cual contiene un salto de línea que debemos eliminar para evitar errores al subirlo a nuestro gestor de contenidos.

❋ Puntos Limpios

Los recursos disponibles en esta sección son:

➲ Punto Limpios (Fijos y Móviles):

Este dataset está disponible en formato CSV, por lo que el procedimiento a seguir para normalizarlo ha sido el mismo que el descrito en la sección anterior para Parques y Jardines: creamos el CSV desde Google Drive y procedemos a ajustar el formato de las columnas de Latitud y Longitud con la utilidad ***Buscar y sustituir.***

El inconveniente de este dataset está en la mitad de los registros correspondientes a los **Puntos Limpios Móviles**. Estos registros no poseen coordenadas al tratarse de una serie de paradas del vehículo itinerante.

El caso es que se nos ocurrió que podríamos señalarlos también en el mapa en lugar de mostrar una pantalla con una simple lista de paradas. Nos parecía interesante la idea de además de leer la dirección donde el camión realiza la parada, poder ver su localización junto con su horario de servicio.

Para poder implementar esta funcionalidad, fue necesario una serie de pasos:

- Extraemos los detalles de los horarios que son comunes para todas las paradas del campo ***DESCRIPCIÓN*** y lo introducimos en la columna ***HORARIO***. De esta manera ***DESCRIPCIÓN*** solo contiene la lista de paradas (separadas por el carácter #) por dirección junto con los detalles particulares de su horario.

- Los campos ***LATITUD*** y ***LONGITUD*** que se encontraban vacíos, contienen ahora una lista de coordenadas que se corresponden con las paradas descritas en el campo ***DESCRIPCIÓN*** y en el mismo orden (también se encuentran separadas por el carácter #). La ubicación de las paradas se ha obtenido manualmente, buscando las direcciones en Google Maps. Estamos hablando de alrededor de unas 5 paradas para cada Punto Limpio Móvil.

Con estos cambios, ahora los Puntos Limpios Móviles también son ubicables en un mapa si se procesan correctamente las cadenas de texto que contienen los campos anteriormente citados.

Como cambios adicionales, usamos el mismo separador (“#”) en las columnas ***HORARIO*** y ***TRANSPORTE*** para identificar los saltos de línea.

➲ Contenedores de ropa:

Este dataset viene dado en un fichero XLS de Microsoft Excel. En este caso ha sido necesario corregir algunas cabeceras y hacer dos tareas con las coordenadas: Transformarlas de ED50 a WGS84 y obtener las coordenadas geográficas en *Latitud* y *Longitud*, ya que sólo estaban incluidas las proyecciones UTM.

Para la transformación del sistema de proyección utilizamos el P.A.G. (Programa de Aplicaciones Geodésicas) del I.G.N. (Instituto Geográfico Nacional) que permite realizar conversiones entre los datums ED50 y ETRS89. No tiene la posibilidad de convertir a WGS84 pero recordamos que las diferencias entre WGS84 y ETRS89 son casi inapreciables.

Este programa permite realizar conversiones a partir de las coordenadas contenidas en ficheros, por lo que el proceso se pudo agilizar fácilmente. Concretamente, le proporcionamos un fichero tabulado con las columnas correspondientes a las coordenadas UTM de los contenedores en ED50 y le pedimos que nos devolviese las coordenadas UTM en ETRS89 y además, también en coordenadas con grados decimales (***LATITUD*** y ***LONGITUD***), que era realmente en las que nosotros estábamos interesados. Con los resultados, actualizamos las columnas UTM con la nueva proyección y añadimos las nuevas ***LATITUD*** y ***LONGITUD*.**

❋ Suministros Limpios

Los recursos sobre Estaciones de Suministros de Combustibles Limpios estaban disponibles en una hoja de cálculo de Microsoft Excel (formato XLS). Esto facilita la tarea de creación del CSV, ya que pudimos aprovechar la estructura de tabla, aunque como ya ocurrió con datasets anteriores, fue necesario realizar unos ajustes adicionales: por un lado, las cabeceras, y por otro, en las coordenadas.

Para resolver el primer problema, ajustamos las cabeceras usando el formato y estilo que declaramos al principio de este capítulo: Eliminamos los espacios, abreviamos los nombres y los escribimos en mayúsculas para identificarlos como cabecera. Esto es importante para la lectura de los objetos JSON que devuelve la API, ya que una cabecera con espacios podría dar problemas a la hora de recuperar las columnas.

En cuanto a las coordenadas, es necesario solucionar un par de inconvenientes: En primer lugar, en el fichero XLS original los campos ***Latitud*** y ***Longitud*** vienen desglosados en grados (º), minutos (') y segundos ('). Nosotros necesitamos que se encuentren solamente en grados decimales para su tratamiento con Google Maps. Además, las coordenadas geográficas usaban el datum ED50, por lo que también requería su transformación a WGS84.

En un principio, pensamos en utilizar de nuevo la calculadora geodésica del Instituto Geográfico Nacional para la transformación de las coordenadas de ED50 a ETRS89 (que como recordamos, para nuestros propósitos, son muy similares), pero por razones que desconocemos, la calculadora no realiza conversiones entre datums cuando las coordenadas están dadas en latitud y longitud. Sin embargo, si se le proporcionan coordenadas en sistema UTM si las convierte (como ya hicimos con los contenedores de ropa). El principal problema era que, en este caso, sólo un tercio de los registros disponían de coordenadas en proyección UTM, así que esta solución quedaba descartada.

Finalmente, considerando el pequeño tamaño del fichero, decidimos ajustar las coordenadas manualmente. Nos ayudamos de una calculadora disponible en la web www.asturnauta.com que permite insertar coordenadas desglosadas y las sitúa en un mapa de Google Maps. Las coordenadas proyectadas sobre el mapa de Google aparecían ligeramente desviadas debido a que no se encuentra en el mismo datum, pero permite al usuario hacerse una idea bastante aproximada de la ubicación real de la estación de suministro. Una vez localizadas, tan solo debíamos pinchar en el mapa y obtener las coordenadas, esta vez, en el datum correcto.

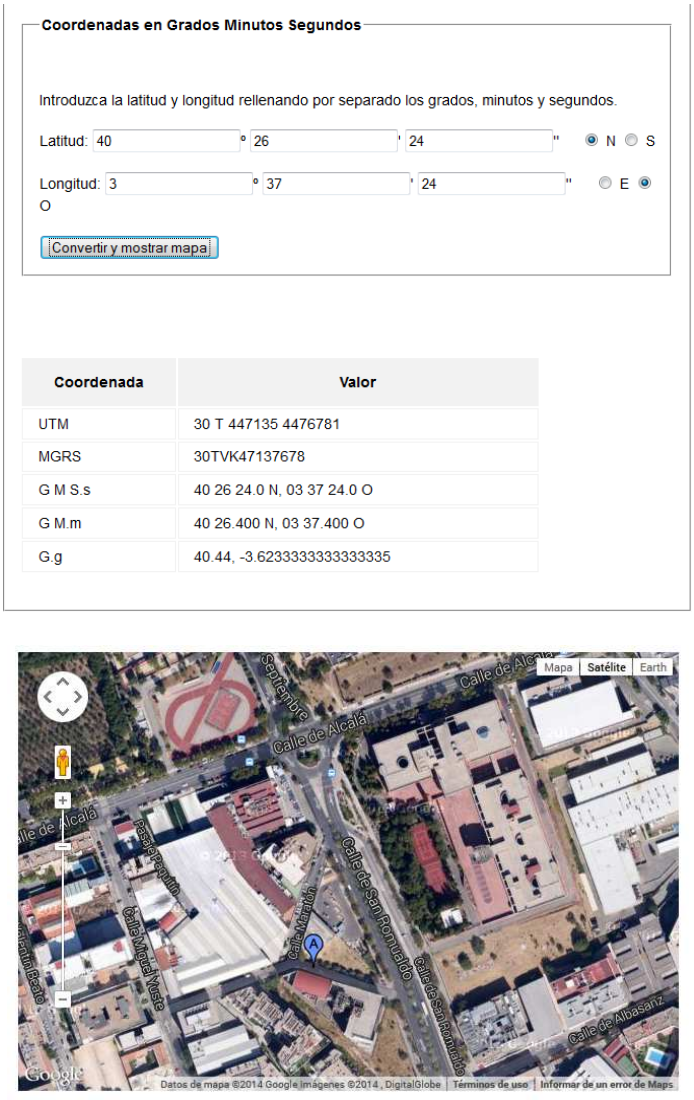


Figura 4.28 – Calculadora de coordenadas de la web Asturnauta

❋ Bicicletas y Motos

Los recursos disponibles para Bicis y Motos en las apps son:

- Aparcamientos para bicis / Reservas Moto
- Áreas de descanso
- Avanza Bici / Avanza Moto
- Calles tranquilas
- Vías ciclistas

Todos estos datasets tienen dos problemas a tratar: están disponibles en formato ShapeFile (*.shp) y el datum usado es el ED50 (recordemos que necesitamos que estén en WGS84). Por lo tanto, debíamos transformar las coordenadas del ShapeFile al datum correcto y luego transformar la información del SHP a CSV. Estos fueron las datasets que más trabajo nos ha dado a la hora de normalizarlos, dado que el formato era totalmente desconocido hasta el momento y además requerían las transformaciones de coordenadas para que no aparecieran desviados en el mapa de Google Maps.

Para resolver el problema de la transformación de coordenadas, fue necesario utilizar la suite ArcGis Desktop y concretamente el programa ArcMap que permite el tratamiento de los ShapeFiles. Esta herramienta posee todo tipo de utilidades para trabajar con mapas, aunque nosotros solo hemos utilizado los métodos para automatizar la transformación del sistema de proyección. Los pasos detallados para llevar a cabo esta tarea se encuentran en el *Anexo I - Transformar coordenadas de un SHP*.

En cuanto a convertir esta información en un CSV se puede realizar de dos maneras. Existe un método sencillo y rápido, que se puede hacer siempre y cuando el sitio web experimental SHP Escape (http://shpescape.com) esté disponible. Está página permite transformar un SHP a un GeoJSON o a Tablas Dinámicas de Google (*Google Fusion Tables*) que se encuentran en fase experimental. Nos decantamos por la opción de las tablas, ya que además de estar preparadas para trabajar con datos geográficos (concretamente en KML, como Google Earth), permite la exportación de dichas tablas en formato CSV que, casualmente, coincide con nuestro objetivo. Otro punto a favor de las tablas dinámicas, es que disponen de una vista de mapa en la que se proyectan los datos de la tabla, lo que permite la comprobación directa de los datos en un mapa de Google Maps.

Si el sitio web está disponible, tan solo hay que subir el fichero ZIP que contiene el proyecto entero con el SHP (sin subcarpetas) y después exportar la tabla dinámica obtenida a formato CSV.

En caso de que el sitio web no esté disponible, serán necesarios unos pasos adicionales: Usaremos otro sitio web experimental, pero esta vez de ArcGis. Se trata de un convertidor de datos geográficos (http://converter.mygeodata.eu) que realiza transformaciones entre ficheros SHP, KML, KMZ, GPX, GeoJSON, etc., además de ser capaz de tratar datos geográficos vectorizados o rasterizados. Con esta herramienta, conseguimos transformar el SHP a KML, que como recordamos, es un formato admitido en las tablas dinámicas de Google. Por lo tanto, el último paso sería crear una nueva tabla dinámica a partir del KML obtenido y esta a su vez, exportarla a formato CSV.

Esta herramienta de ArcGis también permite convertir a CSV, pero el fichero generado no incluye la columna ***geometry*** que es la que contiene las coordenadas del recurso, la obtención de dicha columna y su contenido son nuestro objetivo.

Las instrucciones concretas para la transformación de SHP a CSV se pueden encontrar en el *Anexo II - Transformar SHP a CSV*.

❋ Aparcamientos

Los recursos disponibles en esta sección son:

➲ Aparcamientos para coches:

Estos recursos se encuentran divididos en dos categorías:

- Aparcamientos Públicos + Mixtos
- Aparcamientos Residentes (P.A.R.) + Mixtos

Estos datasets están disponibles en el mismo formato que los de Parques y Jardines, por lo que el proceso a seguir para la normalización fue similar al de dicho dataset: Ajustar el formato de las columnas ***LATITUD*** y ***LONGITUD***. No fue necesario ninguna transformación de coordenadas dado que ya venían en WGS84, aunque sí completar algunas coordenadas que no estaban incluidas. Las localizamos de forma manual usando Google Maps dado que no eran más de diez.

Otro ajuste a destacar en estos dos ficheros es la distribución de la columna ***DESCRIPCIÓN*** entre otras disponibles en el dataset; así, copiamos las plazas de cada aparcamiento en la columna ***EQUIPAMIENTO*** y los textos referentes a los horarios en su columna correspondiente. Por último, identificamos los saltos de línea de todas las columnas con el carácter **#**

➲ Aparcamientos para motos:

Los estacionamientos para motos también fueron proporcionados en formato ShapeFile. Los pasos a seguir fueron los mismos que con los datasets de la sección de **Bicicletas** salvo que esta vez omitimos el paso de la transformación de coordenadas, ya que, este SHP ya estaba en proyección WGS84. Tan solo fue necesario convertirlo a KML con la herramienta online de ArcGis, obtener su tabla dinámica y a partir de ella, su CSV. Después, completamos unos diez registros a los que les faltaban las coordenadas.

Sin embargo, el proceso no fue tan sencillo: una vez subido el fichero CSV al gestor de contenidos, comprobamos que la codificación de caracteres del fichero no era adecuada. Los acentos y caracteres especiales del castellano (como la letra "ñ") no se mostraban correctamente, por lo que fue necesaria su conversión a **UTF-8**. Esto solucionó este problema, pero a su vez introdujo otro.

Los ficheros CSV generados por Google Drive, vienen dados en **UTF-8 sin BOM** [74] (*Byte Order Marker,* en castellano, *Marca de Orden de Bytes*). Esta codificación es igual que **UTF-8**, pero sin un carácter de control (invisible) al comienzo del fichero. En concreto, se trata del carácter `0xFEFF` que significa *zero-width no-break space (*en castellano, espacio no-separable de ancho-cero).

Por alguna razón que desconocemos (aunque intuimos que tiene que ver con la codificación usada por los programas de ArcGis [75]), cuando codificamos el CSV en UTF-8 (es decir, con BOM), los acentos y símbolos se corrigen, pero aparece este carácter al principio del fichero. Decimos que desconocemos la razón, porque no se deberían corregir los acentos cuando incluimos el BOM, puesto que ambas codificaciones son idénticas y se diferencian únicamente en la inclusión del carácter de control.

La aparición del carácter resulta un problema para el identificador de la primera columna, puesto que se concatena con él. Para solucionarlo, es necesario incluir una columna vacía al principio del CSV que no se vaya a usar, ya que no se puede referenciar correctamente. Por esta razón, cuando se trabaja en proyectos web o en sistemas Linux, no se utilizan ficheros que contengan BOM (al contrario que en Microsoft Windows).

La alternativa es dejar la codificación en UTF-8 sin BOM, pero entonces todos los acentos y símbolos especiales puede no mostrarse correctamente dependiendo de los editores: en un Bloc de Notas o Notepad++ se muestran bien, mientras que en Microsoft Excel o en el visor web que tiene el gestor de contenidos de nuestro Open Data no.

➲ **Aparcamientos para bicis:**

Esta sección pertenece a los datasets de **Bicicletas** pero se incluye aquí porque encaja con las que la sección actual de estacionamientos. Las instrucciones de normalización son explicadas anteriormente.

No obstante, el dataset en concreto requirió alguna modificación adicional: ajustamos como en ocasiones anteriores las columnas ***LATITUD*** y ***LONGITUD,*** cambiamos el tipo y formato de algunas columnas por *texto sin formato* y sustituimos los valores numéricos de la columna ***MODELO*** por su descripción en texto, ya que nos pareció que proporciona más información. Para obtener la correspondencia entre valor numérico y descripción utilizamos el Mapa de la Bici disponible en la web del Ayuntamiento de Madrid [76].

❋ Áreas de Prioridad Residencial

Este dataset posee el formato menos estructurado de todos, puesto que nos han proporcionado unos ficheros PDF en los que se aparecían imágenes de los mapas en los que se encerraban las APR. Para su inclusión en las aplicaciones fue necesario crear unos ShapeFiles con ArcMap que nos permitieran obtener las coordenadas de dichas áreas sobre el mapa, así como sus calles de acceso libres.

Después, transformamos los SHP a CSV y fusionamos toda la información en dos ficheros. Estos archivos tan solo tendrán una columna en la que indicarán a que APR se refieren y otra con la lista de coordenadas. Los ficheros CSV finales son:

- APR Calles Libres: Almacena las calles de acceso libre del APR
- APR Áreas: Contiene los puntos del polígono que delimitará la zona APR.

❋ Extras: Herramienta para transformar XML a CSV

Para apoyar el proceso de normalización, también hemos implementado un programa escrito en lenguaje ***Prolog*** al que es posible proporcionarle un fichero XML para convertirlo en CSV.

Esta herramienta puede ser útil en aquellos casos en los que el fichero CSV no existe o, el publicado oficialmente por el Ayuntamiento no está codificado conforme a como lo expusimos en la sección de formatos. Usando este software no es necesario realizar el ajuste de formato para las columnas con coordenadas geográficas. Sin embargo, en los casos que la información se reorganizó entre las columnas (como en los aparcamientos de coches), será necesario aplicar manualmente tal nueva distribución de datos.

4.7 - Protocolo de comunicación

En este último apartado hablaremos de cómo realizar las consultas al Open Data desde las apps. El método en sí, es rápido, sencillo e idéntico para todas las aplicaciones salvando algunas diferencias mínimas, por lo que los ejemplos de código aquí mostrados para la aplicación Mapa de Recursos 2.0, casi se podrían copiar en el resto.

Como ya mencionamos en el apartado dedicado al código fuente de la app, existe una clase denominada **DBConnection** que es la encargada de gestionar las conexiones con el Open Data. Dicha clase se incluye como privada de la actividad Mapa, dado que es en ella donde se realiza la recuperación de los datos para después mostrarlos. La estructura mínima que dicha clase debería tener es:

- La clase **DBConnection**, debe heredar de una clase del *core* de Android denominada *AsyncTask* [77], que nos facilita la ejecución de tareas en segundo plano. Esto es debido a que la recuperación de los datos es una tarea larga, por lo que debe realizarse en un hilo en segundo plano y nunca en el hilo **`main(),`** ya que conduciría a la congelación de la interfaz de usuario, dando la sensación de que la aplicación se ha *colgado*.

- Entre los atributos, se declararán una serie de constantes de tipo *String* en las que se almacene las URLs en las que hacer las consultas. De esta manera, cualquier cambio en las URLs o en los datos es rápidamente actualizable en la app.

- La clase debe contener al menos 2 métodos comunes para todas las conexiones:

    - **connectToUrl():** Ejecuta el proceso de conexión con el servidor

    - **obtainData():** Ejecuta el siguiente paso después de la conexión, que consiste en la recepción de los datos solicitados.

- El resto de atributos y métodos que se incluyan, serán auxiliares para el tratamiento de datos concretos de cada aplicación. En el caso de Mapa de Recursos V2, podemos encontrar el cálculo de la distancia con la fórmula de Haversine y el ordenamiento de los recursos en base a esa distancia mediante Quicksort.

Vamos a analizar estos puntos detalladamente:

❋ Consultas de datos en segundo plano

Para lograr este efecto, se hereda de la clase **AsynTask<T,U,V>,** que permite crear hilos secundarios en los que ejecutar código y, además, mostrar el progreso de dicha tarea al usuario a través de la interfaz gráfica. Los parámetros adicionales de la clase son:

- **T** es el tipo de datos que se le pasarán al método que arranca la tarea en segundo plano. (`execute(T)`)
- **U** es el tipo de datos que se le pasarán al método encargado de actualizar el progreso de la tarea.
- **V** es el tipo de datos que se devolverá al final hilo.

Así tenemos la siguiente estructura inicial. Los métodos que aparecen son los que habría que implementar con nuestras instrucciones particulares:

```
private class DBConnection extends AsyncTask<Void,Integer,Boolean> {

    protected void onPreExecute() {

    protected Boolean doInBackground(Void... params) {

    protected void onProgressUpdate(Integer... values) {

    protected void onPostExecute(Boolean result) {

    protected void onCancelled() {

}
```

Figura 4.29 – Esquema principal de la clase DBConnection

Con la estructura de la figura 4.29 definimos una tarea en segundo plano a la que no le introduciremos ningún tipo de datos de entrada, su progreso lo representaremos mediante números y devolveremos un booleano al final de la ejecución para saber si la tarea se llevó a cabo con éxito. Los cinco métodos que aparecen están sobrecargados (*Override*), y son los que ofrece la clase para simplificar la creación de tareas asíncronas. Alguno de ellos es obligatorio que esté sobrecargado, mientras que otros son opcionales.

- **Obligatorios:**
  - **`onPreExecute()`:** Aquí se introduce el código a ejecutar justo antes de lanzar el hilo secundario. En nuestro caso es donde lanzamos el diálogo que indica que se están descargando los datos desde el Open Data.

```
@Override
protected void onPreExecute() {

    dataLoadProgress.show();

}
```

Figura 4.30 – Ejemplo simple de OnPreExecute()

  - **`doInBackground()`:** Devuelve un valor de tipo V (*boolean* en nuestro caso) y le entran valores de tipo T (*void* en nuestro ejemplo). Aquí se sitúa el código central de la tarea en segundo plano. Este método lo analizaremos con detalle más adelante.

  - **`onPostExecute()`:** Le entran valores de tipo V con el resultado de la operación devuelto por **doInBackground()** (*boolean* en nuestro caso.). En este método se encuentra el código a ejecutar después de completar la tarea asíncrona. En particular, preparamos los recursos para ser mostrados en el mapa siempre y cuando la tarea se haya completado con éxito y a continuación ocultamos el cuadro de diálogo que indica que la descarga está en progreso.

```
@Override
protected void onPostExecute(Boolean result) {

    if (result)
        setUpFirstResourcesOnMap();

    dataLoadProgress.dismiss();
    connectionAlive = false;

}
```

Figura 4.31 – Ejemplo simple de OnPostExecute()

- **Opcionales:**
    - `onProgressUpdate()`: Le entran valores de tipo U, los cuales sirven para actualizar el progreso de la tarea.
    - `onCancelled()`: Si se permite la cancelación de la tarea asíncrona, se puede incluir aquí el código necesario para revertir acciones o volver a a un estado anterior. Aunque nosotros no lo hemos usado, esto se puede valorar para cada app.

❋ API de llamadas unificada

El siguiente punto sería rellenar los atributos de la clase con constantes de tipo *String* que contuvieran las URLs necesarias para realizar las consultas. Un ejemplo que muestra esto para un par de recursos en nuestra aplicación sería:

```
// ATRIBUTOS - CONSULTAS SQL AL OPEN DATA
// ************************************************

private final String sqlQueryURL = "http://www.opendatamadriducm.com/api/action/datastore_search_sql?sql=";

private final String parksQuery = "SELECT * FROM \"87bda271-31fd-4f03-ba58-3ab1f1203631\"";

private final String publicCarParkingsQuery = "SELECT * FROM \"aa8de62d-8b4c-47e0-b58c-809626c58ab6\"";

private final String residentsCarParkingsQuery = "SELECT * FROM \"da1b07b4-27d3-41a8-b170-543abe4a553f\"";
```

Figura 4.32 – Ejemplo de consultas simples para los Parques y los Aparcamientos de Coches

En la figura 4.32 podemos apreciar el uso de las consultas SQL empotradas (*Embedded SQL*). En este caso se ha usado la consultas más sencillas que existen: la que devuelve todos los registros de una tabla concreta (`SELECT * FROM <tabla>`). Sin embargo, es posible incluir cualquier consulta SQL que sea sintácticamente correcta.

Para probar las consultas se puede utilizar cualquier navegador web, aunque recomendamos una herramienta denominada POSTMAN. Es una extensión del navegador Google Chrome que sirve precisamente para depurar APIs REST. Permite introducir una URL (la de la consulta) y visualizar la respuesta devuelta por el servidor debidamente formateada.

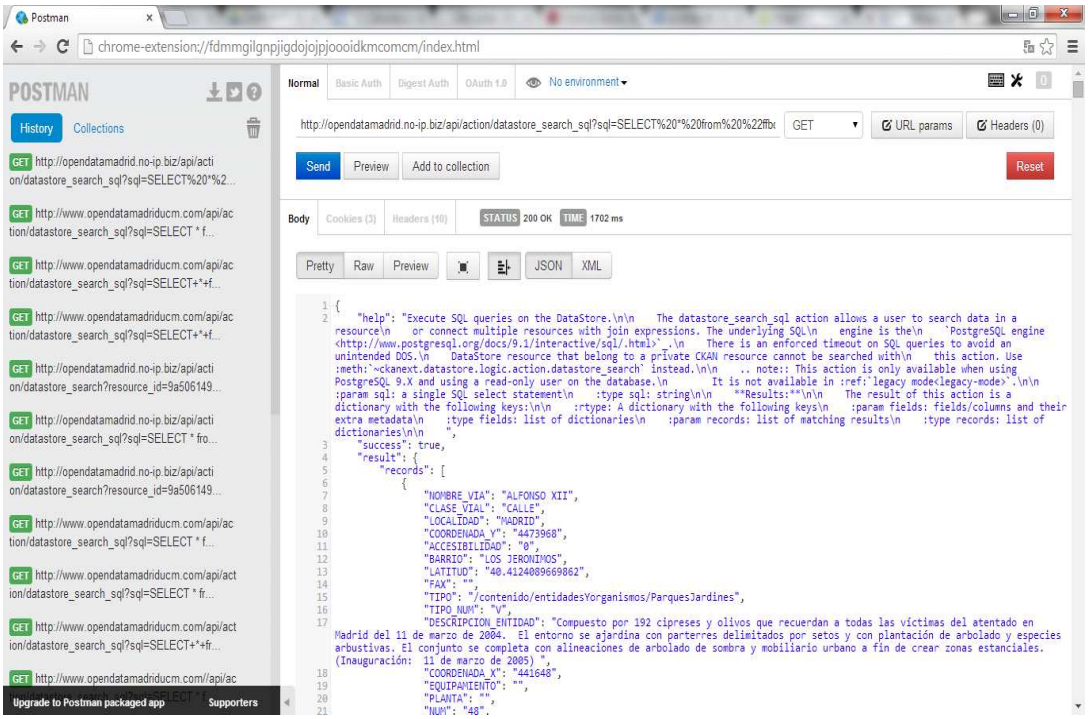


Figura 4.33 – Ejemplo de prueba de una consulta con POSTMAN REST-Client

❋ El método doInBackground()

En este método se debe llamar en el orden dado a:

1. `connectToUrl()`
2. `obtainData()`
3. Ejecutar un método auxiliar que trate el JSON concreto recibido.

Analicemos estas tres fases con más detenimiento:

1. **Método connectToUrl():**

   Este método se encarga de establecer la conexión con el servidor mediante el protocolo HTTP. Para ello, comienza codificando la URL a la que realizar la petición en formato UTF-8, que es el que maneja nuestro sistema Open Data como dijimos en el capítulo tres.

A continuación se configuran los parámetros de conexión, como el *connection time out* o tiempo que pasará sin que obtengamos respuesta del servidor para considerar fallida la conexión. En nuestro caso hemos establecido en doce segundos dicho tiempo.

```
// CONEXION CON DB
// ********************************************************************
InputStream is = null;

// OpenDataAPI : Devuelve todo el dataset en un JSON
String url = "";
try {
    url = sqlQueryURL + URLEncoder.encode(query, "UTF-8");
    Log.d("URL", url);
} catch (UnsupportedEncodingException e2) {
    // TODO Auto-generated catch block
    e2.printStackTrace();
}

try {
    // Establecemos tiempo maximo para conectar (12 seg)
    HttpParams httpParameters = new BasicHttpParams();
    HttpConnectionParams.setConnectionTimeout(httpParameters, 12000);

    // Creamos la conexion HTTP con los parametros anteriores
    HttpClient httpClient = new DefaultHttpClient(httpParameters);
    HttpGet getRequest = new HttpGet(url);
    HttpResponse response = httpClient.execute(getRequest);
    StatusLine responseStatus = response.getStatusLine();
    int statusCode = responseStatus.getStatusCode();
    if (statusCode == 200) {
        HttpEntity entity = response.getEntity();
        is = entity.getContent();
    }
}
catch(Exception e) {
    e.printStackTrace();
    runOnUiThread(new Runnable() {

    return false;
    // En esta captura de excepción podemos ver que hay un problema con la
    // conexión e intentarlo más adelante.
}
```

Figura 4.34 – Código del método que realiza la conexión al Open Data

La figura 4.33 muestra el código concreto para conectar con el Open Data, donde ***sqlQueryURL*** y ***query***, serian parámetros de entrada al método y harían referencia a alguno de los atributos que definimos como constantes (las consultas). El ***statusCode*** = 200 indica que la conexión se realizó satisfactoriamente, es entonces cuando recogemos la respuesta devuelta.

2. **Método obtainData():**

   Una vez recibida una respuesta (un JSON), podemos proceder a extraer la información y a convertirla a *String*. Esto servirá para el posterior procesamiento del JSON. El código del método sería el siguiente:

```
// CAPTURA DEL JSON EN BRUTO
// ********************************************************************************

String received = "";

try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(is,"UTF-8"),8);
    StringBuilder sb = new StringBuilder();
    sb.append(reader.readLine() + "\n");
    String line="0";

    while ((line = reader.readLine()) != null)
        sb.append(line + "\n");

    is.close();
    received = sb.toString();

}
catch(Exception e) {

    runOnUiThread(new Runnable() {

    return false;
}

if (connector.isCancelled())
    return false;
```

Figura 4.35 – Código del método que procesa la respuesta del servidor (extrae el JSON)

   Al final de estas instrucciones, ya tendremos nuestro JSON listo para procesar en la variable *String* llamada ***received.***

3. **Procesamiento del JSON**

   Este método es particular para cada dataset, dado que habrá que analizar qué columnas del CSV queremos procesar. El formato JSON (acrónimo de *JavaScript Object Notation*) es un formato ligero para el intercambio de datos en el que estos se definen como pares Clave-Valor de la siguiente manera:

```
{ "Clave" : "Valor" }
```

Los objetos JSON pueden estar compuestos de varios JSON o incluso de arrays de JSON. He aquí un ejemplo para un JSONArray de dos posiciones:

```
{“Clave1”: [{“Clave2”,”Valor2”},{“Clave3”,”Valor3”}]}
```

Además de los elementos anteriores, se pueden encontrar otros tipos más básicos en los valores:

i. ***Number***: Los valores representan números decimales.

ii. ***String***: Secuencia de cero o más caracteres Unicode encerradas entre comillas dobles. Ejemplo: “ cadena ”.

iii. ***Boolean***: Contiene valores *true* o *false*

iv. ***Null:*** Representa un valor vacío.

Durante la sección dedicada a la normalización de los datasets, propusimos una serie de reglas a cumplir por todos los datasets que se cargaran al sistema Open Data. Entonces, establecimos que el tipo de todas las columnas del CSV debía ser ***texto sin formato***. Esto se hizo para evitar tener que tratar con tantos tipos diferentes a la vez y simplificarlos todos a dos: *String* o *JSONArray*.

Para acceder a todos los campos, el *core* de Android provee unas librerías que facilitan el procesamiento de los JSON. Tan solo hay que realizar llamadas a unos métodos para recuperar un valor asociado a una clave solicitada. Para ilustrar un ejemplo, mostraremos el código en el que se procesa el JSON de los contenedores de ropa.

Hay que destacar, que cuando nuestro sistema Open Data devuelve los JSON, siempre hay que empezar recuperando la clave “**results**” y a continuación el array con clave “**records**” que es el que contiene los registros; en este caso, los diferentes contenedores de ropa. A partir de ahí, se irán solicitando claves al JSON (que se corresponden con las columnas del CSV) para ir recuperando los valores.

La figura 4.35 muestra el código que lleva a cabo el análisis de la información recibida:

```
// ANALISIS DEL JSON
// ********************************************************************************************

String  center,
        district,
        address;

double  latitude,
        longitude;
int     distance;

// Creamos un nuevo array para guardar los parques
currentResources = new ArrayList<BasicResource>();
int rec = 0;
try {

    // Encajamos en un objeto JSON lo recibido
    JSONObject jsonObject = new JSONObject(received);

    // Cogemos la etiqueta "result" que tiene los resultados que nos interesa
    JSONObject result = jsonObject.getJSONObject("result");

    // Despues cogemos el array de "records" que tiene la etiqueta "results"
    // que es donde estan realmente los resultados que queremos
    JSONArray jsonArray = result.getJSONArray("records");

    for(int i = 0; i < jsonArray.length(); i++){

        jsonObject = jsonArray.getJSONObject(i);

        center = jsonObject.getString("CENTRO");

        address = jsonObject.getString("DIRECCION");
        district = jsonObject.getString("DISTRITO");


        latitude = longitude = -1.0;
        try{
            latitude = Double.parseDouble(jsonObject.getString("LATITUD"));
            longitude = Double.parseDouble(jsonObject.getString("LONGITUD"));
        }
        catch (Exception e){
            Log.d("JSON", "Fallo en recurso numero: " + i);
            Log.d("JSON", jsonObject.toString());
        }

        // Calculamos la distancia actual al recurso
        LatLng referencePos;    // Nuestra posicion de referencia
        LatLng itemPos;     // Posicion del parque actual
```

```
        // Si es Mi Ubicacion, cogemos la ultima conocida
        if (myLocationBased){
            Location l = mLocationClient.getLastLocation();
            referencePos = new LatLng(l.getLatitude(),l.getLongitude());
        }
        else
            referencePos = new LatLng(addressCord[0], addressCord[1]);

        // Asignamos las coordenadas del parque
        itemPos = new LatLng(latitude,longitude);

        // Calculamos finalmente la distancia
        distance = computeDistanceHaversine(referencePos, itemPos);


        ClothingContainer cc = new ClothingContainer(latitude,
                                                     longitude,
                                                     distance,
                                                     center,
                                                     district,
                                                     address);

        currentResources.add(cc);
    } catch (JSONException e1) {
        Log.d("JSON", "Fallo en recurso numero: " + rec);
        runOnUiThread(new Runnable() {

        return false;
    }

    return true;
```

Figura 4.36 – Código del método en el que se procesan las claves concretas del JSON.

Una vez ejecutado este código, tendremos un array en la app (**`currentResources)`** donde hemos ido guardando la información del JSON en objetos **`ClothingContainer,`** los cuales ya están listos para trabajar con ellos.

Para finalizar, daremos el código necesario para ejecutar la tarea asíncrona que hemos preparado a lo largo de esta sección. En nuestro caso, el código se sitúa en el método **`OnCreate()`** de la actividad Mapa.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_map);

       . . .

    // Preparar el dialogo de proceso de descarga de datos
    dataLoadProgress = new ProgressDialog(MapActivity.this);
    dataLoadProgress.setProgressStyle(ProgressDialog.STYLE_SPINNER);
    dataLoadProgress.setMessage("Descargando datos desde el Open Data...");
    dataLoadProgress.setCancelable(false);

    // Seteamos ya el mapa para que el Thread lo tenga disponible
    setUpMapIfNeeded();

    connector = new DBConnection();
    connectionAlive = true;
    connector.execute();

       . . .
```

Figura 4.37 – Código para iniciar la ejecución de la tarea asíncrona en el método OnCreate()

V. Análisis y Resultados

Durante el siguiente capítulo vamos a repasar brevemente las soluciones creadas en este proyecto de manera que ilustremos su potencial, su funcionamiento y ejemplifiquemos su uso. Comenzaremos hablando de nuestro Sistema Open Data dando unos ejemplos de usos básicos para el público general y después para un usuario administrador del sistema. Queremos remarcar en este punto, que hemos tratado de realizar una simulación lo más fiel posible del portal oficial de Datos Abiertos del Ayuntamiento de Madrid. Esto lo hemos reflejado en que hemos mantenido, en la medida de lo posible, las mismas estructuras y contenidos que existían en los originales.

A continuación revisaremos la aplicación Mapa de Recursos 2.0 para mostrar las aplicaciones y salidas finales del Open Data. Para reforzar esta idea, se decidió crear otro sitio web adicional (independiente del Open Data, puesto que es una aplicación web externa) el cual emula el comportamiento de la app. De esta manera demostraremos que nuestro sistema no solo sirve para aplicaciones destinadas a ejecutarse en terminales móviles.

Para terminar, realizaremos un análisis comparativo entre nuestro Open Data y la solución liberada por el Ayuntamiento en abril de 2014 junto con un análisis de escalabilidad para nuestra propuesta, en la que detallaremos qué configuraciones son necesarias contratar en función de la demanda de servicio que se quiera brindar.

5.1 Ejemplos de uso de la propuesta de Open Data para el Ayuntamiento de Madrid

En esta sección vamos a ilustrar el funcionamiento del Open Data mediante unos sencillos ejemplos.

Los conjuntos de datos que contiene nuestro Open Data son idénticos a los que se encuentran en el catálogo oficial de datos abierto del Ayuntamiento de Madrid con algunas excepciones que describimos en el capítulo 4 durante el proceso de normalización. En cualquier caso, nuestra intención es emular el Open Data del Ayuntamiento de Madrid junto con algunas mejoras.

5.1.1 Cliente

A continuación vamos a ilustrar las funcionalidades que puede usar un usuario regular en nuestro Open Data:

❋ Página de inicio:

Figura 5.1– Página principal de nuestro portal Open Data

Esta es la página de inicio que nos encontramos al entrar con un navegador. Podemos distinguir los siguientes elementos que describimos a continuación:

- En la parte superior encontramos el menú que será igual para todo el sitio web.

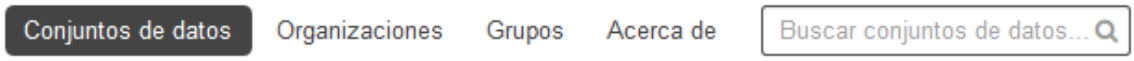


Figura 5.2– Secciones principales de nuestro portal Open Data

- En el pie de página encontramos una serie de enlaces con información que, al igual que el menú anterior, están presentes en todas las secciones del sitio web. Destacamos que se puede cambiar el idioma de toda la página con el menú desplegable de la figura 5.3.

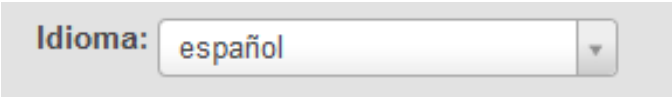


Figura 5.3– Selector de idioma de nuestro portal Open Data

- En el inicio tenemos unas estadísticas que nos dirán el número de conjuntos de datos publicado, organizaciones, grupos y elementos relacionados (que suelen ser las aplicaciones desarrolladas usando estos datos).

Figura 5.4– Estadísticas de nuestro portal

- Por último, destacar que en la parte superior existe un botón para *Iniciar Sesión*. En nuestro caso, hemos desactivado la opción de crear nuevos usuarios, puesto que no es necesario disponer de un perfil de usuario para acceder a los datos. Por lo tanto, este acceso está destinado al uso exclusivo de los administradores del sitio web, desde donde podrán eliminar o añadir conjuntos de datos, grupos, organizaciones, etc.

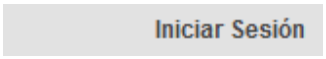


Figura 5.5– Botón de inicio de sesión para administradores

✱ Catálogo de conjuntos de datos:

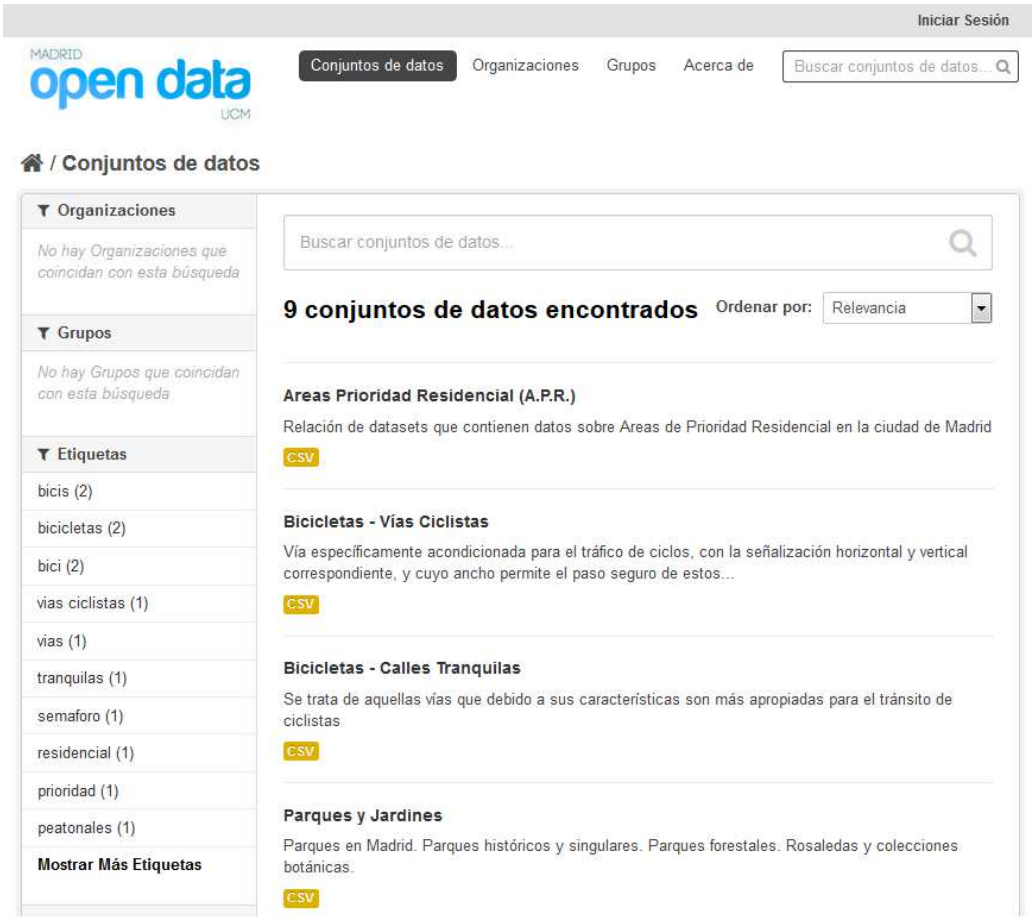


Figura 5.6 – Buscador y filtro de conjuntos de datos

En la figura 5.6 se muestra la página en la que se encuentra el buscador de conjuntos de datos. Se muestran en un listado, en los que aparecerá el nombre, una pequeña descripción y en qué formatos se encuentran los datos. Es posible mejorar la búsqueda utilizando los filtros que aparecen en el menú de la izquierda.

❋ Un conjunto de datos:

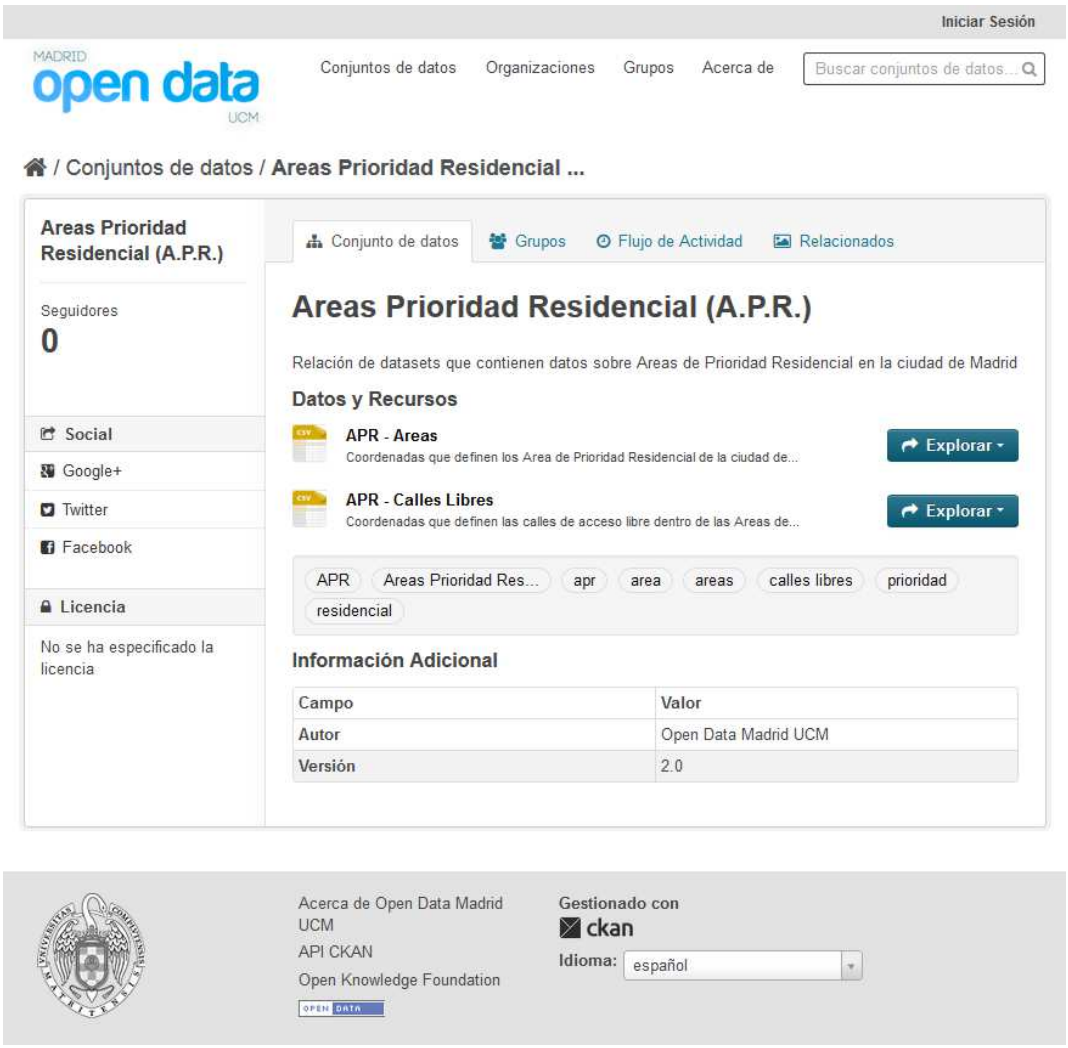


Figura 5.7 – Un conjunto de datos con sus ficheros disponibles

Entrando dentro de un conjunto de datos podremos ver información ampliada del mismo, así como los diferentes datos que contiene, los formatos en los que están disponibles dichos datos, a que organización pertenecen, que tipo licencia poseen, etc. Si añadimos "*.rdf*" o "*.n3*" al final de la URL, se muestra toda esta información en formato RDF.

➲ Si pulsamos en el botón *Explorar* en un dato, aparecerá un menú en el que podremos elegir si queremos descargarlo o pre-visualizarlo.

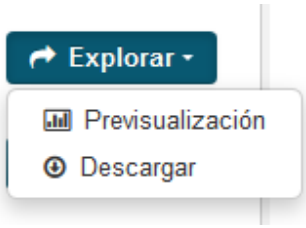


Figura 5.8 – Operaciones sobre un conjunto de datos del Open Data

❋ Pre-visualización de un dato:

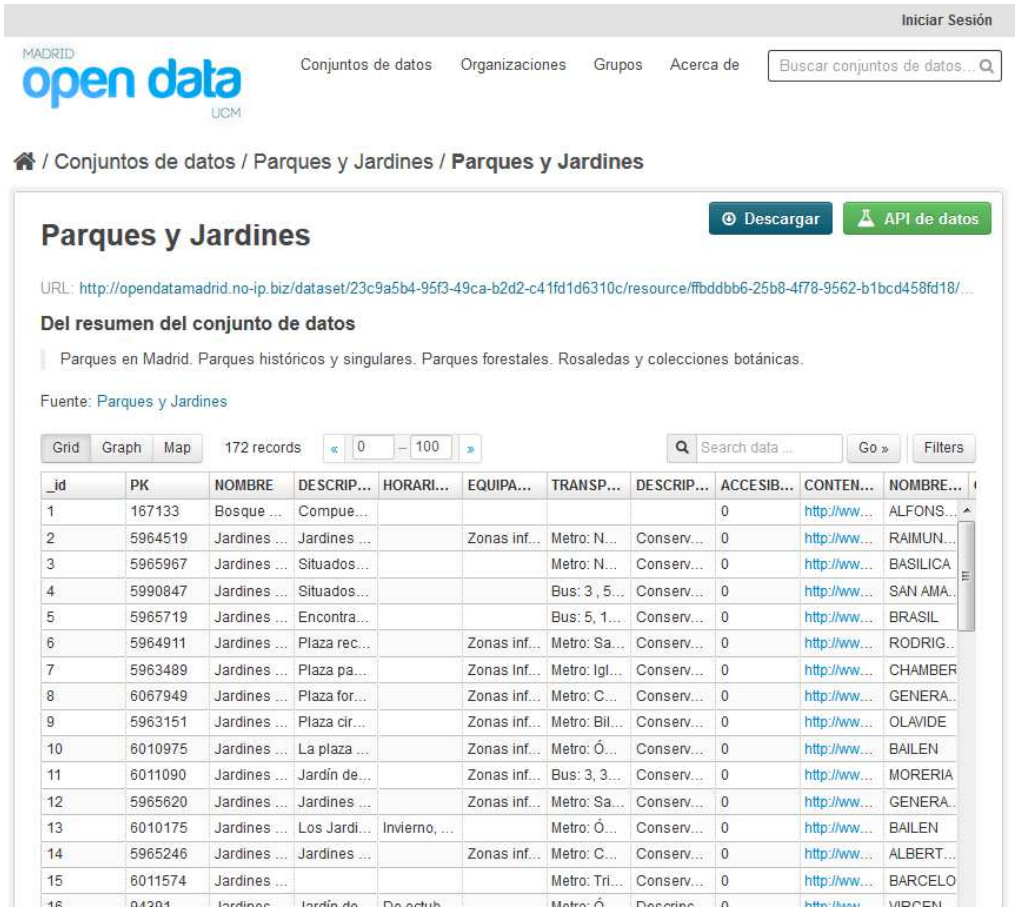


Figura 5.9 – Pre-visualización en forma de tabla de un conjunto de datos

Dependiendo del formato que sea, se pre-visualiza de una manera u otra. Si son *XLS* o *CSV*, será en forma de tabla tal como se muestra en la figura 5.9; si son texto plano, se verá el texto que contienen; si es un PDF, se abrirá el visor de PDF. Por último, si son imágenes, se mostrarán las mismas. En los casos que el formato no corresponde con ninguno de los anteriores, no se pre-visualizará nada.

Cuando la pre-visualización es en forma de tabla (ficheros CSV y XLS), es posible filtrar en ella o usar el buscador para localizar datos. Además tendremos unas cuantas opciones más, que resumimos a continuación:

- Se habilitará la *API REST*, lo que permitirá que terceras personas puedan consultar los datos con las llamadas a la *API*. Para la creación, actualización y eliminación de datos vía API es necesario una API Key que no se hace pública en ningún momento, por lo que estas operaciones están reservadas exclusivamente a administradores. Si pulsamos el botón *API de Datos* se abrirá una ventana explicando, con unos ejemplos, como se usa.

Figura 5.10 – Instrucciones de uso de la API REST para un conjunto de datos

- Si pulsamos en el botón *Graph*, podremos crear gráficos con los datos:

Figura 5.11 – Gráfico creado a partir de valores de un CSV o XLS

- Por último si pulsamos en *Map*, se nos abrirá un mapa y si tenemos entre los datos la latitud y la longitud se situaron en el mapa los diferentes recursos.

Figura 5.12 – Mapa situando las coordenadas geográficas contenidas en un CSV o XLS

❋ Organizaciones:

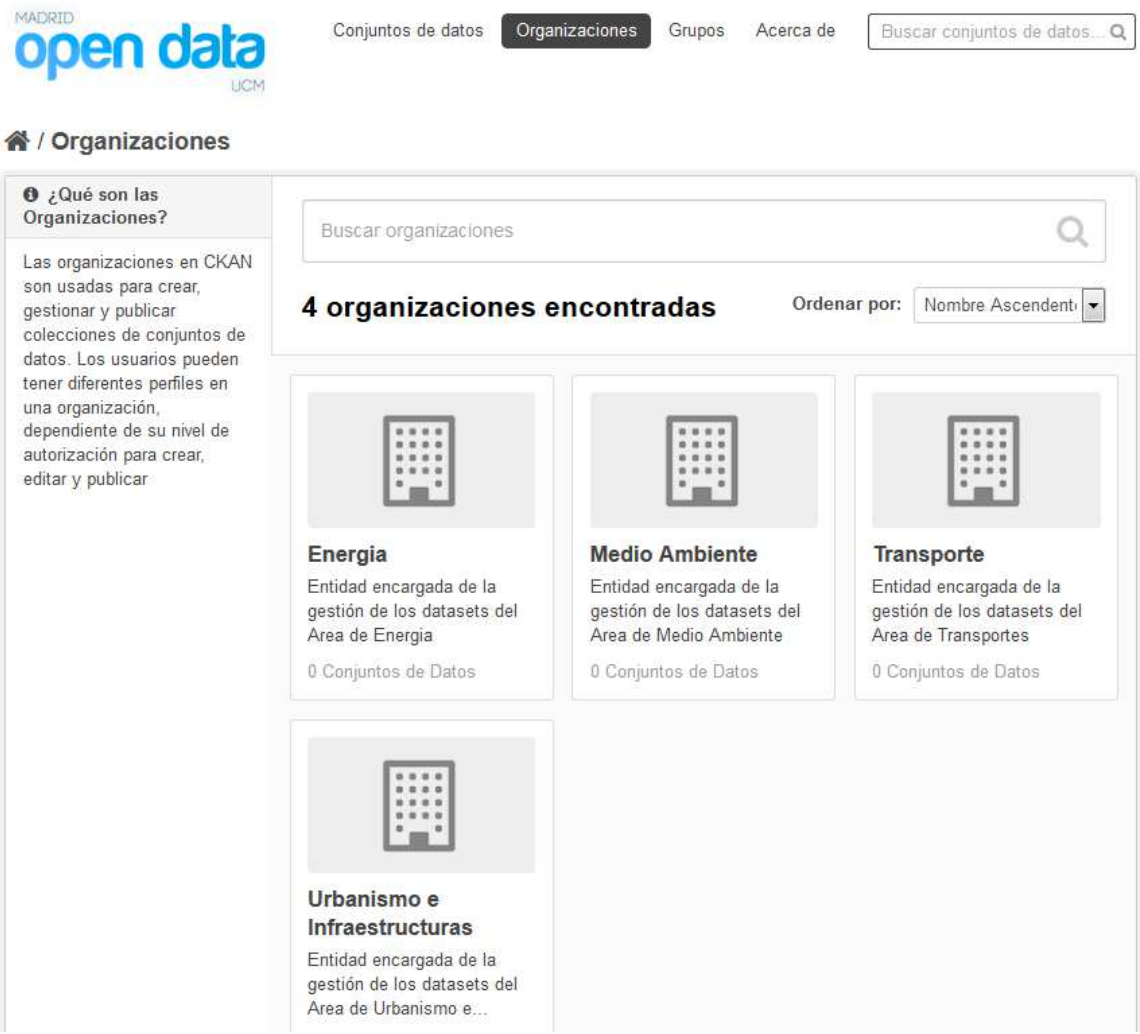


Figura 5.13 – Organizaciones que mantienen conjuntos de datos en nuestro Open Data

En esta página podremos ver las organizaciones que existen, pudiendo ordenarlas o usar el buscador para encontrar una concreta. Las organizaciones sirven para agrupar los conjuntos de datos en diferentes secciones. En nuestro caso, las secciones representan diferentes áreas de gobierno del Ayuntamiento como Medio Ambiente, Energía, etc.

- Si seleccionamos una nos llevará a la página que aparece en la figura 5.14 en la que se muestra una descripción y los diferentes conjuntos de datos que contiene.

Figura 5.14 – Conjuntos de datos correspondientes a la organización de Medio Ambiente.

5.1.2 Administrador

A continuación vamos a ilustrar las diferentes opciones que puede usar un administrador en nuestro Open Data:

❋ Registro:

Figura 5.15 – Iniciar sesión en nuestro Open Data.

Si queremos conectarnos al área de administración, debemos hacerlo a través del registro de usuarios (figura 5.15). Solo los usuarios administradores podrán hacerlo. En caso de precisar usuarios adicionales se tendrá que hacer desde el servidor a través de la consola.

❋ Crear un nuevo conjunto de datos:

Para añadir nuevos conjuntos de datos debemos acceder a la sección *Conjuntos de Datos*. Como ahora somos usuario administrador, en la parte superior del buscador aparecerá un botón para poder agregarlos (figura 5.16).

Figura 5.16 – Conjuntos de datos en vista administrador

Si pulsamos dicho botón aparecerá la siguiente página:

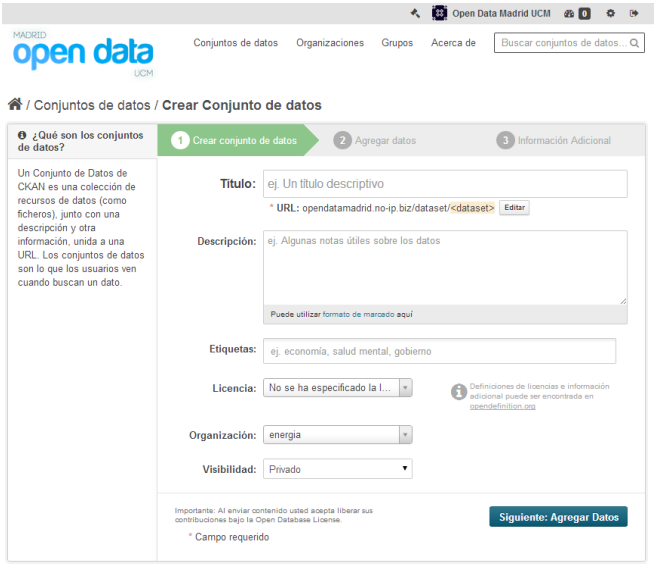


Figura 5.17 – Formulario a rellenar para agregar un nuevo conjunto de datos

Una vez completado el paso anterior, pulsamos en *Siguiente*. Se mostrará la página de la figura 5.18

Figura 5.18 – Añadir ficheros a los conjuntos de datos

En este paso es donde agregaremos los recursos (ficheros) a los conjuntos de datos. Lo primero que tenemos que distinguir es que podemos subir ficheros que tengamos en nuestra maquina o proporcionar un enlace a otra máquina remota (por ejemplo, otro servidor o sitio web) que indique donde estén almacenados los datos que queremos subir. Recordamos que si subimos un dato en formato *CSV* o *XLS* automáticamente se generará un *API REST* del recurso sin que tengamos que hacer nada.

Además de cargar el fichero, es recomendable completar los campos que aparecen, sobre todo el del formato, ya que ayudaremos a nuestro gestor de contenido a identificar este sin problemas.

Una vez este todo listo, si pulsamos en *Guardar y Agregar otro* podremos añadir otro archivo más. Esto sirve en los casos que queremos cargar el mismo dataset en diferentes formatos. Si pulsamos en *Siguiente* nos aparecerá el último paso que deberemos realizar.

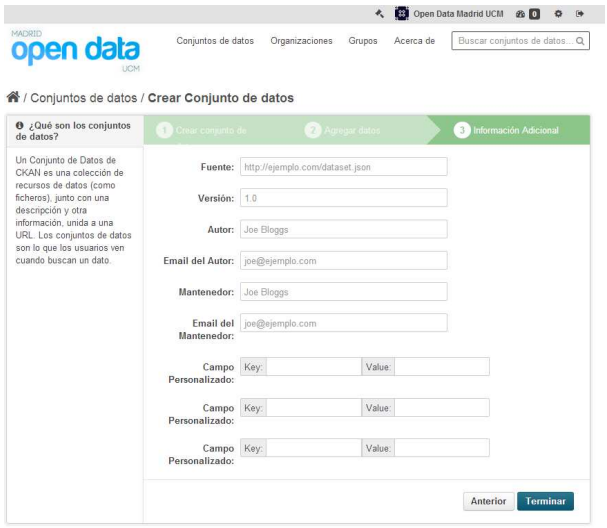


Figura 5.19 – Información adicional de un conjunto de datos

Opcionalmente, se puede añadir más información a los conjuntos de datos como la fuente de los mismos, la versión, los autores, los encargados de su mantenimiento, etc. Para finalizar el proceso y añadir el conjunto de datos al gestor de contenidos pulsamos en *Terminar*

✱ Eliminación de un conjunto de datos:

Buscamos el conjunto de datos que queremos eliminar y pulsamos en el botón *Administrar* (figura 5.20)

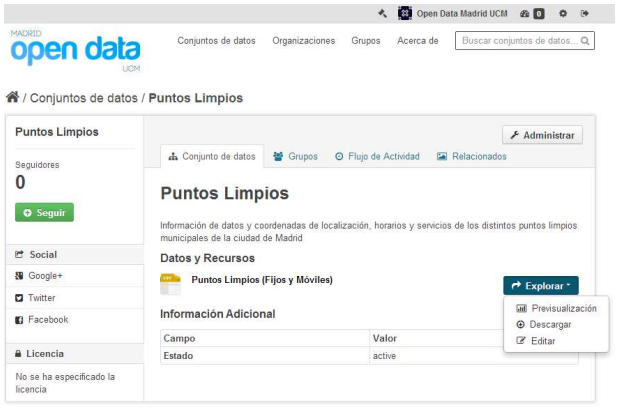


Figura 5.20 – Vista de un conjunto de datos por un administrador

A continuación se abrirá una nueva página donde podremos actualizar los datos o borrarlos (figura 5.21). En la pestaña *Recursos* podremos eliminar o añadir nuevos ficheros al conjunto de datos.

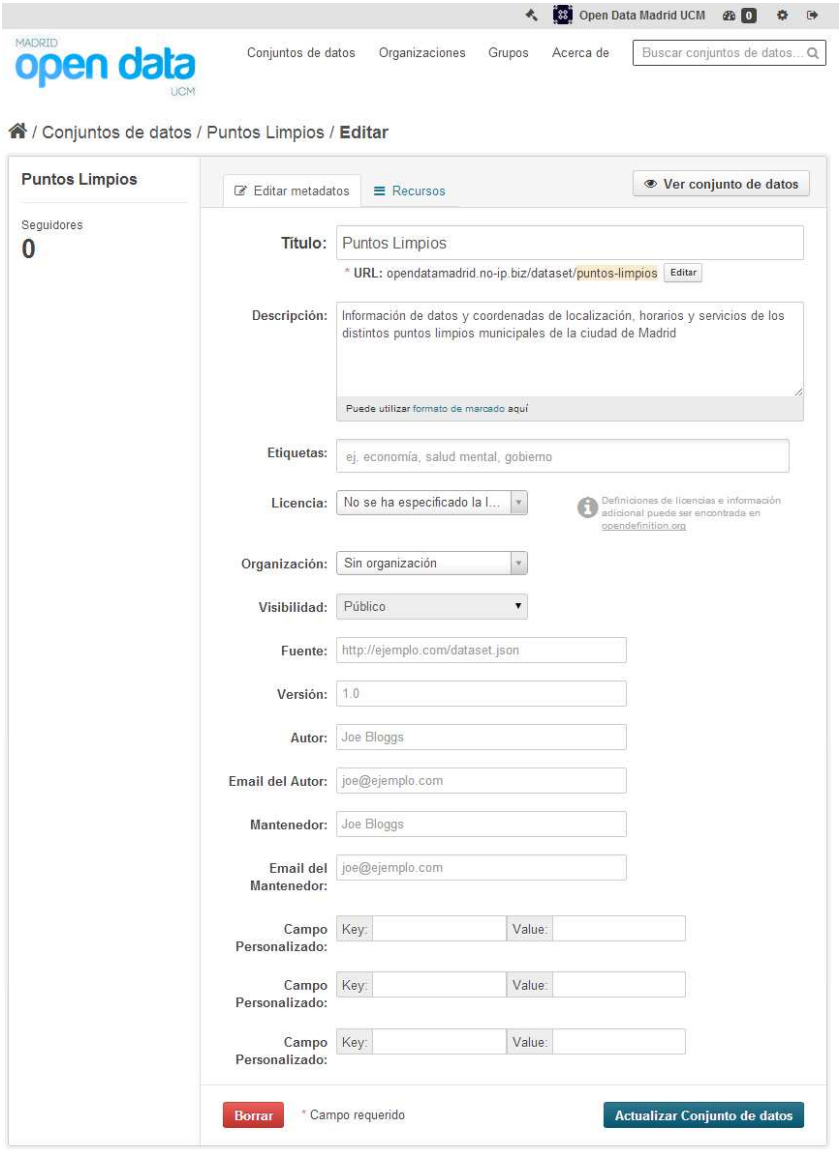


Figura 5.21 – Editando la información de un conjunto de datos

Si seleccionamos *Borrar* el conjunto de datos desaparecerá de la página, pero en realidad no se borrará definitivamente. Para eliminar cualquier rastro es necesario entrar en la papelera. Esto se hace a través de la siguiente URL:

```
www.opendatamadriducm.com/admin/trash
```

Aparecerá la página de la figura 5.22, donde deberemos pulsar en el botón (mal traducido) *Púrguelas todas (para siempre y de forma irreversible)*. Con eso si aseguramos que se eliminarán para siempre.

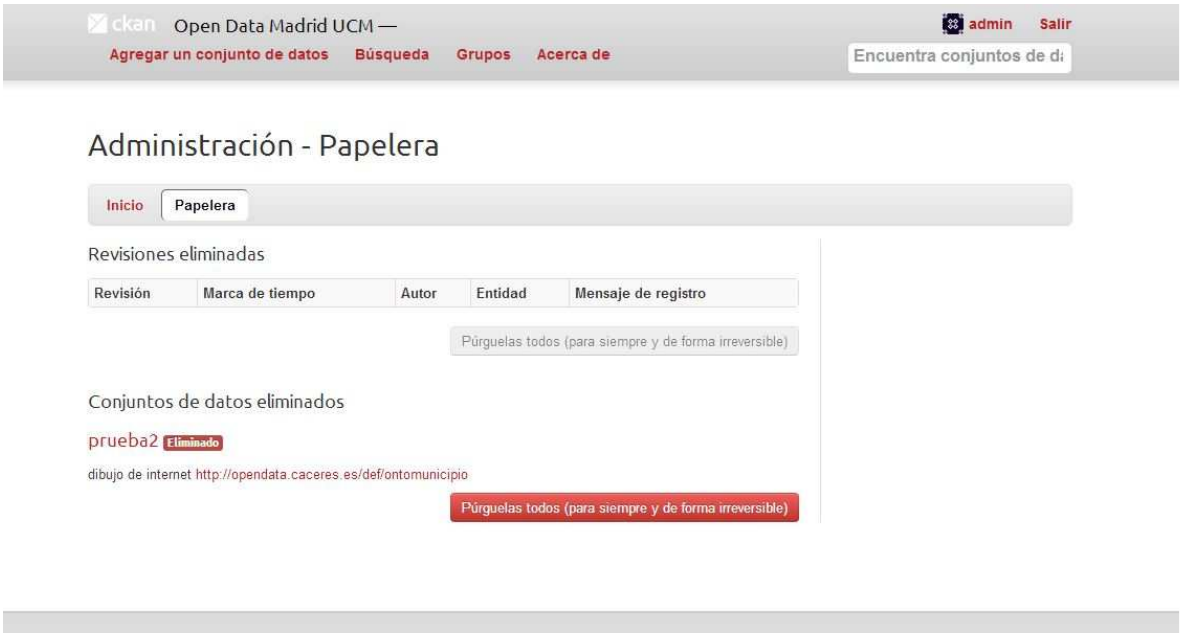


Figura 5.22 – Papelera de reciclaje del Open Data

5.2 Ejemplos de uso y resultados para Mapa de Recursos V2

En esta sección vamos a ilustrar el funcionamiento de la aplicación mediante unos sencillos ejemplos de uso mediante los cuales veremos los cuatro tipos de mapas que podemos encontrar en esta nueva versión de la app.

❋ Menú principal

Una vez se encuentre instalada la aplicación en un terminal Android, si la ejecutamos, la primera pantalla que aparece es el menú principal donde se recogen las 6 categorías principales de la app:

- **Parques y Jardines**
- **Aparcamientos**
  - Coches (Públicos y Residentes)
  - Motos
  - Bicis
- **Puntos Limpios**
  - Fijos
  - Móviles
  - Contenedores de ropa
- **Suministros Limpios**
  - Bioetanol
  - GLP
  - GNC
- **Bicicletas y Motos**
  - B: Área de Descanso
  - B. Avanza Bici
  - B. Aparcabicis
  - B: Calles Tranquilas
  - B: Vías Ciclistas
  - M: Avanza Moto
  - M: Reservas Moto
- **Áreas de Prioridad Residencial**

Figura 5.23 – Menú principal de la app Mapa de Recursos 2.0

Si la sección no contiene subcategorías, se accederá a la pantalla donde se elige que tipo de ubicación se utilizará, sino, aparecerá una pantalla en la que se muestran las subopciones disponibles (figura 5.24).

Figura 5.24 – Subopciones para aparcamientos y para bicicletas

❋ Elección del tipo de ubicación:

En este menú, el usuario decide si quiere utilizar el GPS o las redes móviles para calcular su ubicación actual de forma aproximada o proporcionar una dirección postal. En este último caso, el usuario escribe una dirección y la aplicación le muestra un máximo de 5 posibles resultados a esa dirección. El usuario se deberá decidir por uno. En cualquier de los dos casos, la ubicación elegida sirve para mostrar después los recursos más cercanos a dicha localización.

Figura 5.25 – Elección de la ubicación del usuario en Mapa de Recursos 2.0

❋ Conexión con el Open Data

Como explicamos en la sección dedicada al protocolo de comunicación, la tarea de conectarse y descargar los datos no es inmediata y es un proceso delicado en el que muchas cosas pueden fallar. Por ello, durante las solicitudes al servidor, mostramos la siguiente pantalla en la que se informa al usuario de que se están obteniendo los datos.

Figura 5.26 – Diálogo de progreso que indica que se están descargando datos

❋ Mapas de recursos

Una vez concluida la recepción de los datos y la decodificación del JSON, se muestran los datos sobre el mapa. Existen cuatro tipos de mapa diferentes:

➲ **Mapa basado en puntos.**

Es un mapa en el que aparecen marcadores resaltando las ubicaciones solicitadas. Si se pulsa sobre ellos, se puede ampliar la información acerca de la localización del marcador. Este es el mapa que se usa en todos los recursos excepto para los puntos limpios móviles, las vías ciclistas, las calles tranquilas y las Áreas de Prioridad Residencial.

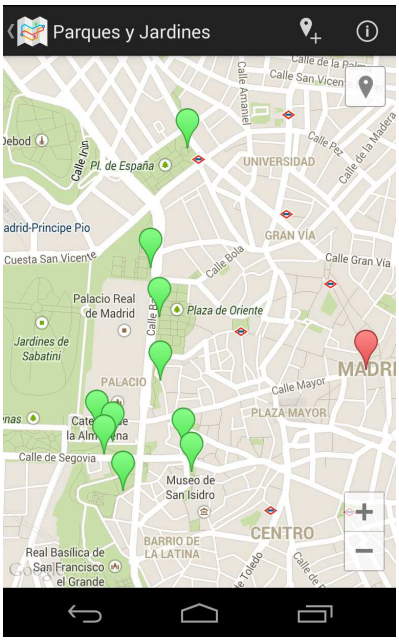


Figura 5.27 – Mapa de recursos que muestra puntos

➲ **Mapa basado en líneas y puntos.**

Este es un caso especial exclusivo de los puntos limpios móviles. Existen 16 y están repartidos uno en cada distrito de la ciudad. Se decidió mostrar las paradas de dichos servicios itinerantes utilizando marcadores que al pulsarlos, amplían la información sobre la parada. También se han unido las paradas del servicio de cada distrito con polilíneas. Los círculos semitransparentes alrededor del marcador indican que la ubicación es aproximada, dado que el camión estacionará en aquel sitio que se encuentre disponible.

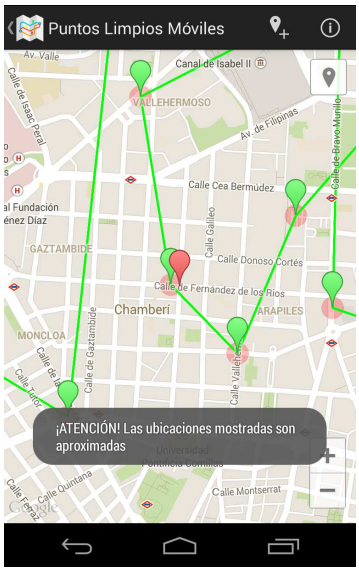


Figura 5.28 – Mapa de recursos que muestra puntos unidos con líneas

➲ Mapa basado en líneas

Son los casos de las Calles Tranquilas y las Vías Ciclistas, que se representan en el mapa mediante polilíneas que indican dichas calles y vías dedicadas a los usuarios de bicicletas

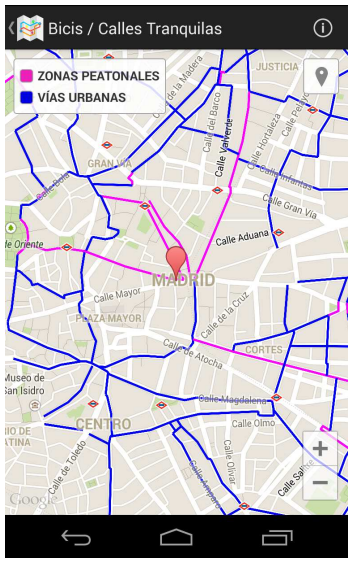


Figura 5.29 – Mapa de recursos que muestra calles representadas con líneas

➲ **Mapa basado en polígonos y líneas.**

Este es el último caso especial y corresponde con las APR. Es necesario dibujar un polígono para indicar el área correspondiente al APR y después, marcar las calles con acceso libre con polilíneas.

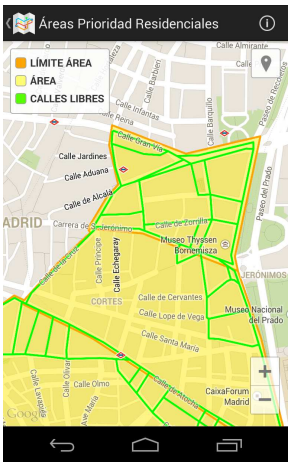


Figura 5.30 – Mapa de recursos que muestra Áreas de Prioridad Residencial

❋ Información de los recursos

Todos los recursos disponen de una sección en la aplicación en la que se pueden consultar toda la información general referente a ellos. Como ya dijimos, si el mapa muestra puntos, es posible pulsar sobre ellos para ampliar la información del recurso en concreto.

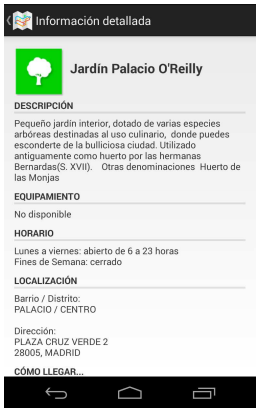


Figura 5.31 – Información particular de un recurso

5.2.1 Extras: Página web Mapa de Recursos usando Open Data

Una vez desarrollado nuestro proyecto Open Data sobre CKAN con los conjuntos de datos proporcionados por el Ayuntamiento de Madrid, decidimos desarrollar la aplicación Mapa de Recursos para Android como primera aplicación de los recursos que ofrece nuestra plataforma de datos abiertos. Sin embargo, creímos conveniente presentar en nuestro proyecto otra aplicación de nuestro sistema de datos abiertos, para mostrar que son varias las posibles iniciativas que podemos llevar a cabo a la hora de crear nuevos servicios que usen un Open Data.

Por un lado, podemos encontrarnos en la situación de no disponer de nuestro dispositivo móvil o bien de no tener acceso a internet con él y, por otro lado, podemos encontrarnos con un usuario que, por el motivo que sea (suponiendo que hoy en día cualquier persona dispone de un *smartphone* o *tablet,* lo cual no es del todo cierto), no disponga del sistema operativo en su dispositivo que soporte cierta aplicación, o simplemente no tenga la app descargada porque no tiene más memoria en disco o porque no desea hacerlo pues su intención es darle un uso momentáneo o temporal.

Además, una persona que no tenga conocimientos para desarrollar aplicaciones móviles, sí podría tener conocimientos para plasmar su iniciativa sobre otro medio igual o más utilizado que las aplicaciones móviles: una página web.

Por todo ello, decidimos desarrollar la versión web de la app Mapa de Recursos.

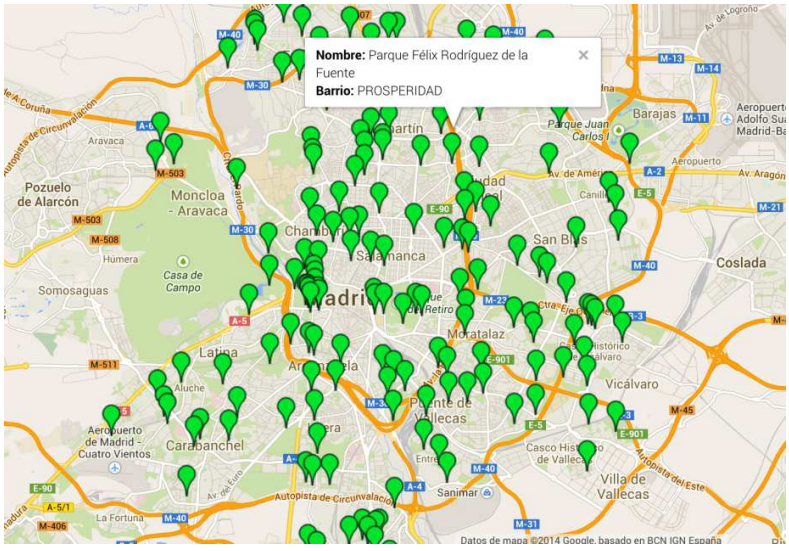


Figura 5.32 – Página web que emula el comportamiento de la app Mapa de Recursos

❋ Tecnologías usadas

Para desarrollar la web nos hemos utilizado ningún software específico, ni tampoco ningún *framework* sobre el que maquetar la web. Hemos utilizado editores de código corrientes (Sublime Text principalmente), y programado la página en HTML combinado con PHP y *Javascript*. A continuación les describimos estos lenguajes:

- **HTML**: Sus siglas provienen de ***H**yper**T**ext **M**arkup **L**anguage* (en castellano, Lenguaje de Marcas de HiperTexto), y es que no se trata de un lenguaje de programación propiamente dicho, sino de un lenguaje de marcado. Podemos definir un lenguaje de marcas como una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación. De hecho, se trata del lenguaje de marcado más extendido, y es fundamento del World Wide Web. Es además un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, por lo que podemos concluir con que HTML es el lenguaje con el que se definen las páginas web.

- **PHP:** Influenciado fuertemente por C y C++, PHP es un lenguaje de programación de uso general del lado del servidor, originalmente diseñado para el desarrollo web de contenido dinámico. La **programación del lado del servidor** es una tecnología que consiste en el procesamiento de una petición de un usuario mediante la interpretación de un script en el servidor web para generar dinámicamente páginas HTML como respuesta.

  PHP fue uno de los primeros lenguajes de programación del lado del servidor que se pudo incorporar directamente en un documento HTML. El código debe ser interpretado por un servidor web que soporte PHP para que genere la página web resultante. Desde su creación en 1995, el grupo PHP lo sigue actualizando. Este lenguaje forma parte del software libre publicado bajo la licencia PHP, funciona en la mayoría de los servidores web, así como en casi todos los sistemas operativos y plataformas sin ningún coste.

- **JavaScript**: Tradicionalmente usado sobre páginas web HTML, JavaScript es un lenguaje de programación interpretado, es decir, está diseñado para ser ejecutado por medio de un intérprete, a diferencia de los lenguajes de programación compilados. Además, se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

  Es utilizado principalmente en su forma en la **parte del cliente** (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque también se utiliza para la

parte del servidor. Además, tiene un significativo uso externo a la web, sobre documentos PDF, widgets, entre otros. Tiene una sintaxis similar a C, y adopta nombres y convenciones de Java. Sin embargo, Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes. Todos los navegadores modernos interpretan el código JavaScript incorporado en las páginas web.

❋ Funcionamiento de la web

Hemos desarrollado una primera versión de la web sobre una estructura sencilla en HTML, queriendo mostrar una página simple pero muy limpia y no menos funcional. El protagonista de la página web no es otro que el mapa centrado en la Comunidad de Madrid sobre el que se mostrarán los recursos que se están consultando.

Para ello, hemos recurrido una vez más al servicio de mapas ofrecido por Google, y hemos obtenido una nueva API Key para esta página web. Después, utilizando JavaScript, incorporamos código a nuestra web y los scripts necesarios para el correcto funcionamiento del servicio de mapas (Google Maps API v3). De ahí en adelante, todo el manejo y edición de mapas sobre la web se realiza en JavaScript, que es el lenguaje utilizado por Google para este servicio.

Una vez puesto el mapa en funcionamiento, necesitamos hacer uso de nuestros datos abiertos. Gracias a nuestro sistema Open Data desarrollado con CKAN, podemos realizar fácilmente consultas sobre los conjuntos de datos que disponemos. Estas consultas, realizadas en el lenguaje de programación PHP, nos devolverán una respuesta en formato JSON con los datos obtenidos desde nuestro Open Data.

Cuando se tiene almacenado en un array los datos del JSON, se procesan de principio a fin para generar los objetos buscados en nuestra consulta. Cada uno de estos objetos será una clase instancia implementada con anterioridad. En nuestro caso, las consultas serán sobre parques, aparcamientos, puntos limpios, entre otros. Cada uno de estos recursos de medio ambiente y movilidad tendrá implementada su propia clase en PHP. De esta manera, conseguiremos tener una estructura clara y ordenada para nuestros objetos en la web, y una vez realizada una consulta, podremos, combinando PHP (para trabajar con nuestros objetos) y JavaScript (para trabajar con el mapa de Google), finalmente mostrar al usuario la localización sobre un mapa y todos los recursos del Ayuntamiento de Madrid solicitados.

5.3 Análisis comparativo

En esta sección vamos a comparar el Open Data que hemos creado nosotros con el del Ayuntamiento de Madrid.

Las principales diferencias las enumeramos a continuación:

- Los dos Open Data usan un gestor de contenidos. El del Ayuntamiento de Madrid usa ***Vignette***, nosotros usamos **CKAN**. *Vignette* no es de código abierto, mientras que CKAN sí, lo cual va en contra de la filosofía de los Open Data.

- En cuanto al gestor de bases de datos ellos usan **Microsoft SQL Server 2005 SP2**, nosotros **PostgreSQL**. Microsoft SQL Server no es código abierto, PostgreSQL sí.

- Los dos utilizan buscadores, el del Ayuntamiento de Madrid, usa ***Autonomy 7.4*** y nosotros ***Solr***. De nuevo *Solr* es de código abierto y *Autonomy* no.

- Ellos usan servidores propios y nosotros usamos servidores en la nube con *Amazon Web Service*.

- Nuestro Open Data tiene una API REST de los diferentes conjuntos de datos, el del Ayuntamiento de Madrid no. Esto implica que a partir de un Open Data sea mucho más costoso desarrollar aplicaciones que reutilicen estos datos.

- Nuestros conjuntos de datos pueden pre-visualizarse en la web, los de ellos no. Se tienen que descargar para poder verlos.

- El Open Data del Ayuntamiento de Madrid tienen cierta cantidad de conjuntos de datos con los formatos incorrectos.

- Los dos tienen los archivos RDF con URIs para la web semántica.

Estas son las principales diferencias que se pueden encontrar entre los dos Open Data, también se ha de destacar que el Open Data del Ayuntamiento de Madrid solo lleva unos pocos meses en funcionamiento, con lo que es de esperar que a lo largo del tiempo irá evolucionando.

5.4 Análisis de escalabilidad

En esta sección vamos a exponer los requisitos hardware que recomienda CKAN. A continuación, analizaremos las diferentes alternativas que podemos elegir y finalmente explicaremos por cual nos hemos decantado.

5.4.1 Recomendaciones hardware CKAN

En la documentación de CKAN se especifican los recursos hardware que ellos recomiendan para un Open Data, distinguiendo entre dos casos que exponemos a continuación:

❋ Open Data para una ciudad sin mucha demanda

Las especificaciones son las siguientes:

- Dos servidores: uno para el sitio web y otro para la base de datos.
- Cada uno de ellos debería contar con, al menos, 2GB de *RAM* y un procesador *dual core* (doble núcleo).
- 80 GB de almacenamiento en cada uno. Sin embargo, esta cantidad será variable y dependerá de la cantidad de conjuntos de datos que se vayan a subir.

❋ Open Data para un país con mucha demanda

Las especificaciones son las siguientes:

- Dos servidores: uno para el sitio web y otro para la base de datos.
- Cada uno de ellos debería contar con, al menos, 8GB de *RAM* y un procesador *quad core* o superior.
- 160GB de almacenamiento en cada uno; cantidad variable que depende de la cantidad de conjuntos de datos que se vayan a subir.

Nuestro Open Data estaría contenido en el caso de una ciudad sin mucha demanda, ya que es para la ciudad de Madrid. Lo único a destacar, es que estas especificaciones hardware debemos tomarlas con precaución. Esto es debido a que podrían ser insuficientes en momentos puntuales, entre otras razones porque la ciudad de Madrid es muy grande y podría tener una demanda considerable.

5.4.2 Alternativas

En esta sección analizaremos las diferentes alternativas disponibles para el servidor que contendrá el Open Data. Y expondremos las ventajas y desventajas que tienen cada uno.

❋ Servidores dedicados

Los beneficios de esta opción es que los servidores los controlas y no dependes de terceros. Aparte, es más seguro, ya que los datos tampoco están bajo el control de un tercero.

La desventaja principal es que se desperdician recursos hardware, ya que se necesitan tener más de los necesarios (hay que tener en cuenta momentos puntuales de aumento de carga). Además, si por alguna razón el Open Data aumenta su popularidad muy rápidamente o se calcula mal el número de personas que lo van a usar, se tendrá un problema, debido a que no se tendrá suficientes recursos hardware con los que hacer frente y, aumentar dichos recursos, no se puede hacer en el momento, sino que podría llevar semanas al necesitar comprarlos e instalarlos.

❋ Amazon Web Service

El principal beneficio de esta opción es que no se necesitan tener contratados más recursos hardware que los que se requieran en cada momento. Si fueran necesarios más, simplemente se aumentarían, lo que puede hacerse en minutos.

Las desventajas son que los servidores no los controla el administrador del Open Data, y que los datos están almacenados en un servidor externo.

A continuación exponemos las diferentes alternativas que tenemos para implementar la arquitectura en los servidores de *Amazon*:

- Usar un único servidor *EC2* con almacenamiento *EBS* para todo el Open Data, es decir, que dicho servidor incluya el sitio web y la base de datos.

- Igual que el punto anterior pero para los objetos estáticos usar *Amazon S3*. Si hay muchos estáticos lograríamos ahorrar algo de dinero.

- Usar dos servidores de Amazon *EC2:* uno para el sitio web y otro para la base de datos. El del sitio web no necesitaría almacenamiento *EBS*. Este almacenamiento sólo es necesario e en el servidor de las bases de datos.

- Igual que el anterior pero conectando, en el servidor de la base de datos, un *Amazon S3* para los estáticos. Así se ahorraría dinero al necesitar menos almacenamiento *EBS*.

Nosotros hemos elegido para nuestro Open Data la primera opción por su sencillez y porque no se espera tener una demanda tan alta como para que no pueda aguantarlo un único servidor; además, a medida que se necesitan más recursos se pueden ir solicitando como explicamos al principio.

5.4.3 Pruebas de estrés

A continuación se exponen unas pequeñas pruebas de estrés realizadas a nuestro Open Data donde llevamos el servidor a su límite. Se tiene que tener en cuenta que estas pruebas se han realizado con el servidor más pequeño que ofrece *Amazon Web Service* y que se podrían mejorar muchísimo estos números si contratamos más recursos hardware.

Hemos usado dos benchmarks: el *ApacheBench* y el *Siege*. Para controlar el uso de recursos del servidor usaremos el comando top de *Linux*.

❋ Pruebas con ApacheBench

Ejecutamos *ApacheBench* con 1000 peticiones y 50 peticiones concurrentes.

```
Server Software:        nginx/1.1.19
Server Hostname:        54.72.3.99
Server Port:            80

Document Path:          /
Document Length:        11209 bytes

Concurrency Level:      50
Time taken for tests:   90.552 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      11613000 bytes
HTML transferred:       11209000 bytes
Requests per second:    11.04 [#/sec] (mean)
Time per request:       4527.580 [ms] (mean)
Time per request:       90.552 [ms] (mean, across all concurrent
requests)
Transfer rate:          125.24 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       43  767  2872.7   49      31102
Processing:   118 3220 1719.8   3968    5795
Waiting:      114 3214 1720.0   3957    5792
```

```
Total:      165 3987 2696.3   4186    31234

Percentage of the requests served within a certain time (ms)
 50%   4186
 66%   4478
 75%   4624
 80%   4766
 90%   5161
 95%   7186
 98%  15203
 99%  15273
100%  31234 (longest request)
```

La *CPU* está al 100% de capacidad en este test y no llenamos la memoria *RAM*. En estos resultados podemos apreciar que las peticiones tardan en responderse de media 4'5 segundos. Esto nos indica que el servidor está bastante al límite debido a que deberíamos esperar 4'5 segundos hasta que se cargara una página del portal.

Ejecutamos *ApacheBench* con 1000 peticiones y 100 peticiones concurrentes.

```
Resultados:

Server Software:        nginx/1.1.19
Server Hostname:        54.72.3.99
Server Port:            80

Document Path:          /
Document Length:        11209 bytes

Concurrency Level:      100
Time taken for tests:   193.187 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      11613000 bytes
HTML transferred:       11209000 bytes
Requests per second:    5.18 [#/sec] (mean)
Time per request:       19318.673 [ms] (mean)
Time per request:       193.187 [ms] (mean, across all concurrent
requests)
Transfer rate:          58.70 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       42   398   1824.7  48      15198
Processing: 2269 18593 9489.8  19536   41661
Waiting:    2265 18587 9489.9  19534   41657
Total:      2320 18991 9886.2  19586   49521

Percentage of the requests served within a certain time (ms)
 50%  19586
 66%  25918
 75%  26670
 80%  27145
 90%  29499
```

```
 95%  34479
 98%  39075
 99%  40582
100%  49521 (longest request)
```

En este caso, el consumo de *CPU* fue intermitente porque el servicio se caía continuamente. Aunque no perdimos ninguna solicitud, el servidor se encontraba saturado, haciéndose imposible su uso como demuestra el hecho de que tarda 19 segundos de media en responder.

❋ Pruebas con Siege

Ejecutamos *Siege* durante 60 segundos y 50 peticiones concurrentes.

```
Resultados:

Lifting the server siege...   done.
Transactions:                 686 hits
Availability:                 100.00 %
Elapsed time:                 59.55 secs
Data transferred:             2.08 MB
Response time:                3.42 secs
Transaction rate:             11.52 trans/sec
Throughput:                   0.03 MB/sec
Concurrency:                  39.40
Successful transactions:      686
Failed transactions:          0
Longest transaction:          7.90
Shortest transaction:         0.16
```

En este caso la CPU estaba al 80%-90% de uso y podemos observar que no se pierde ninguna petición con lo que el servidor se encuentra funcionando correctamente.

Ejecutamos *Siege* durante 60 segundos y 100 peticiones concurrentes.

```
Resultados:

Lifting the server siege...   done.
Transactions:                 688 hits
Availability:                 99.42 %
Elapsed time:                 59.28 secs
Data transferred:             2.08 MB
Response time:                6.50 secs
Transaction rate:             11.61 trans/sec
Throughput:                   0.04 MB/sec
Concurrency:                  75.42
Successful transactions:      688
Failed transactions:          4
Longest transaction:          19.79
Shortest transaction:         0.20
```

En este caso la *CPU* estaba al 100% de uso y podemos observar que el servidor se encuentra al límite. Empieza a rechazar peticiones porque no puede tratar tantas.

A la conclusión que llegamos con estos datos es que el servidor no puede tratar más de 50 peticiones concurrentes, superando este límite el servidor empieza a funcionar erráticamente. Aunque deberemos tomar estos datos con precaución, ya que en un uso real el rendimiento será menor. Esto es debido a que han sido realizados por *Benchmark* y estas pruebas no cargan los archivos *CSS* y *JavaScript*. Si el servidor fuera a tener más demanda se deberían contratar más recursos hardware en *Amazon Web Service*.

VI. Conclusiones y trabajos futuros

6.1 Conclusiones

El proyecto que hemos desarrollado durante este año demuestra que con no demasiados recursos es factible crear un Open Data potente, de forma que éste sea muy útil para el ciudadano y pueda ser aprovechado por las nuevas tecnologías de la información y la comunicación (NTIC).

Nuestra plataforma es un Open Data de gobierno (propuesta Open Data de Madrid), destinado al ciudadano y siguiendo los ideales *Open Government* que cada vez más están influenciando a la mayoría de gobiernos del mundo. A través de nuestro Open Data, los habitantes de Madrid tienen a su total y directa disposición los datos que recoge el Departamento de Medio Ambiente y Movilidad del Ayuntamiento, así como la oportunidad de generar un beneficio a través de ellos.

Nuestro proyecto es una clara iniciativa *Smart City* (ciudad inteligente) en Madrid, por lo que destaca en cuanto a su innovación y repercusión en el desarrollo actual de las ciudades. Sin ir más lejos, la Unión Europea ya ha puesto fecha límite a los países que la integran para la puesta en marcha de sus plataformas Open Data y promover así el desarrollo de ciudades inteligentes en nuestra sociedad.

Además, hemos llevado a cabo con ejemplos reales varios de los usos que podemos dar a un Open Data:

- Creación de una API que proporciona al ciudadano una librería de funciones que facilita la consulta y el procesado de los datos publicados.

- Creación de una aplicación móvil que ofrece algún tipo de servicio gracias a los datos publicados en nuestro Open Data.

- Desarrollo de una versión web de dicha aplicación.

Nuestra aplicación para Android, “Mapa de recursos”, es el mejor ejemplo de aplicación “inteligente” y “verde” siguiendo el concepto Smart City, en la que se proporciona información muy completa (geolocalización, información descriptiva, etc.) sobre los diferentes recursos de la ciudad de Madrid que se refieren al departamento de Medio Ambiente y Movilidad.

Además, con la versión web de la aplicación, cumplimos con uno de los principales ideales que defienden los datos abiertos, y es el llegar al mayor número de personas posibles y de la manera más accesible, es decir, a través de Internet.

6.2 Trabajos futuros

El proyecto realizado se podría extender realizando alguna de las siguientes mejoras o adaptaciones:

- Mejorar el plugin *DataPusher* de CKAN, o desarrollar uno nuevo, para poder decodificar conjuntos de datos que estén en formatos diferentes a XLS y CSV.

- Adaptar el resto de aplicaciones Android que el grupo G-Tec ha desarrollado junto con el el departamento de Medio Ambiente y Movilidad del Ayuntamiento de Madrid (Camino seguro al cole, Recycla.me, etc.), para que funcionen a través de los datos contenidos en nuestro Open Data y puedan utilizar su API y demás funcionalidades.

- Realizar la versión web equivalente al resto de aplicaciones del grupo G-Tec, obteniendo de nuevo los datos a través de nuestro Open Data, de forma análogo a como hemos hecho con la aplicación “Mapa de recursos”.

- Incluir en el Open Data, además de los datasets de Medio Ambiente y Movilidad, otros conjuntos de datos de diferentes áreas o temáticas que sirvan para aumentar las posibilidades de uso del sistema.

- Y lo más interesante, investigar ideas innovadoras para desarrollar nuevas aplicaciones que funcionen bajo nuestro Open Data y que generen un beneficio importante en el camino de Madrid hacia una *Smart City*.

Referencias

[1] BBVA [https://www.centrodeinnovacionbbva.com/innovation-edge/21-big-data]

[2] Wikipedia - Open Data [http://en.wikipedia.org/wiki/Open_data]

[3] USA Open Data [http://data.gov]

[4] UK Open Data [http://data.gov.uk]

[5] Wikipedia - Open Government [http://en.wikipedia.org/wiki/Open_government]

[6] Lathrop, Daniel; Ruma, Laurel, eds. (February 2010). Open Government: Transparency, Collaboration and Participation in Practice. O'Reilly Media. ISBN 978-0-596-80435-0.

[7] OGP (La Alianza para el Gobierno Abierto) [http://www.opengovpartnership.org]

[8] Fundación C-TIC – Smart Cities [http://datos.fundacionctic.org/etiquetas/smart-city/]

[9] Wikipedia – Smart City [http://en.wikipedia.org/wiki/Smart_city]

[10] Portal de datos abiertos del Gobierno de los Estados Unidos de América [http://data.gov]

[11] Portal de datos abiertos de la Unión Europea [http://open-data.europa.eu]

[12] Emitter - Portal del Area de Medio Ambiente de Canadá [http://emitter.ca]

[13] Antonio Ferrer-Sapena, Fernanda Peset y Rafael Aleixandre-Benavent - Acceso a los datos publicos y su reutilizacion: Open Data y Open Government (2011)

[14] Portal de Datos Abiertos del Gobierno de Navarra [http://gobiernoabierto.navarra.es]

[15] Portal de Datos Abiertos del Gobierno de Euskadi [http://opendata.euskadi.net]

[16] Portal de Datos Abiertos de la Empresa Municipal de Transportes (EMT) de Madrid) [http://opendata.emtmadrid.es]

[17] Portal de Datos Abiertos del Ayuntamiento de Madrid [http://datos.madrid.es]

[18] W3.org – Dessign Issues – Linked Data [http://www.w3.org/DesignIssues/LinkedData.html]

[19] Página Grupo Tecnología UCM (G-Tec) [http://www.tecnologiaucm.es]

[20] Documentación oficial CKAN [http://ckan.org/]

[21] Wikipedia - API [http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones]

[22] Wikipedia - REST [http://es.wikipedia.org/wiki/Representational_State_Transfer]

[23] Documentación Python [http://es.wikipedia.org/wiki/Python] y [https://www.python.org/]

[24] Documentación Framework Pylons [http://docs.pylonsproject.org/projects/pylons-webframework/en/latest/]

[25] Documentación WSGI [http://wsgi.readthedocs.org/en/latest/index.html]

[26] Documentación PostgreSQL [http://www.postgresql.org.es/sobre_postgresql]

[27] Documentación SQLAlchemy [http://www.sqlalchemy.org/]

[28] Documentación Apache Solr [http://lucene.apache.org/solr/]

[29] CKAN en GitHub [https://github.com/ckan/ckan]

[30] Wikipedia - Nginx [http://en.wikipedia.org/wiki/Nginx]

[31] Wikipedia - Proxy Server [http://en.wikipedia.org/wiki/Proxy_server]

[32] Documentación *modwsgi* https://code.google.com/p/modwsgi/

[33] Documentación *libpq* http://www.postgresql.org/docs/8.1/static/libpq.html

[34] Documentación plugin *Stats* [http://docs.ckan.org/en/1117-start-new-test-suite/stats.html]

[35] Documentación plugin *Text Preview* [http://docs.ckan.org/en/latest/maintaining/data-viewer.html#text-preview]

[36] Documentación plugin *DataStore* [http://docs.ckan.org/en/latest/maintaining/datastore.html]

[37] Documentación plugin *DataPusher* [http://docs.ckan.org/projects/datapusher/en/latest/]

[38] Documentación plugin *FileStore* [http://docs.ckan.org/en/latest/maintaining/filestore.html]

[39] Documentación Ubuntu [http://www.ubuntu.com/]

[40] Wikipedia - Virtualización [http://es.wikipedia.org/wiki/Virtualizaci%C3%B3n]

[41] Wikipedia - Máquina virtual [http://es.wikipedia.org/wiki/M%C3%A1quina_virtual]

[42] Wikipedia Hypervisor [http://es.wikipedia.org/wiki/Hypervisor]

[43] IBM Hypervisor [http://www.ibm.com/developerworks/ssa/library/l-hypervisor/]

[44] Wikipedia virtualización [x86 http://es.wikipedia.org/wiki/Virtualizaci%C3%B3n_x86]

[45] Documentación *VirtualBox* [https://www.virtualbox.org/]

[46] Wikipedia - dirección IP [http://es.wikipedia.org/wiki/Direcci%C3%B3n_IP]

[47] Wikipedia - IP [http://es.wikipedia.org/wiki/Internet_Protocol]

[48] Wikipedia - DNS [http://es.wikipedia.org/wiki/Domain_Name_System]

[49] Goolgle DNS [https://developers.google.com/speed/public-dns/?hl=es]

[50] Wikipedia computación en la nube [http://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube]

[51] IBM computación en la nube [http://www.ibm.com/developerworks/ssa/websphere/techjournal/0904_amrhein/0904_amrhein.html]

[52] Documentación Amazon EC2 [http://aws.amazon.com/es/ec2/]

[53] Documentación Amazon EBS [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html]

[54]. Documentación Amazon EC2 Security Groups [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html]

[55]. Documentación Amazon EC2 Elastic IP Addresses http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html

[56] Mapa de Recursos (Memoria Oficial)

[57] Android Dashboards [http://developer.android.com/about/dashboards/index.html]

[58] Fragmentación de Android [http://alt1040.com/2014/01/fragmentacion-android]

[59] Google I/O 2011 [https://www.google.com/events/io/2011/]

[60] Google I/O 2012 [https://www.google.com/events/io/2012/]

[61] Dando soporte a diferentes tamaños de pantalla (Android) [http://developer.android.com/guide/practices/screens_support.html]

[62] Google Maps API v2 para Android [http://developer.android.com/google/play-services/maps.html]

[63] Fragments en Android [http://developer.android.com/guide/components/fragments.html]

[64] Wikipedia: CSV [http://en.wikipedia.org/wiki/CSV]

[65] Wikipedia: KML [http://en.wikipedia.org/wiki/XML]

[66] Wikipedia: KML [http://en.wikipedia.org/wiki/KML]

[67] Wikipedia: ShapeFile [http://en.wikipedia.org/wiki/Shapefile]

[68] Wikipedia: Microformato GEO [http://es.wikipedia.org/wiki/Microformato_GEO]

[69] Wikipedia: RDF [http://en.wikipedia.org/wiki/Resource_Description_Framework]

[70] Wikipedia: Portable Document Format [http://en.wikipedia.org/wiki/Portable_Document_Format]

[71] Instituto Geográfico Nacional - Sistemas Geodésicos [http://www.ign.es/ign/layoutIn/actividadesGeodesiaStmagd.do]

[72] Ministerio de Fomento - Real Decreto 1071/2007 [http://www.fomento.gob.es/MFOM/LANG_CASTELLANO/ORGANOS_COLEGIADOS/CSG/ETRS89/]

[73] Instituto Geográfico Nacional - Sistemas Geodésicos (FAQ) [http://www.ign.es/ign/layoutIn/faqgd.do]

[74] BOM - Marca de orden de Bytes - http://es.wikipedia.org/wiki/Marca_de_orden_de_bytes_%28BOM%29

[75] ¿Que es un BOM? – {Sección de Comentarios: Problemas visualización mapas hechos con ArcGis} [http://www.arumeinformatica.es/blog/bom-en-utf-8-que-es/]

[76] Mapa de la Bici de Madrid - Tipos de aparcabicis [http://www.infobicimadrid.es/gis_bicis.htm]

[77] Android Developers - AsyncTask [http://developer.android.com/reference/android/os/AsyncTask.html]

Anexos

Anexo I - Transformar coordenadas en un ShapeFile

❋ Instalación de ArcGis Desktop 10.2.1 para Windows

Accedemos a la web de ESRI (www.esri.com) y nos registramos. Para nuestros propósitos, con la cuenta de evaluación de 60 días es más que suficiente. Comprobar que la cuenta creada dispone del servicio ArcGis Online, ya que a través de él obtendremos los mapas base. Si no es así, actualizar la cuenta para obtener también la evaluación de ArcGis Online.

Una vez hemos completado el registro, accedemos a la sección de productos y seleccionamos ArcGis Desktop 10.2.1 (no confundir con ArcGis Explorer Desktop, el cual no nos sirve ya que no admite ShapeFiles). En nuestro caso hemos seleccionado la versión en castellano, por lo que los comandos de este tutorial se proporcionarán en dicho idioma.

File Description	Documentation	Download Size	
Microsoft .Net Framework 3.5 SP1			Microsoft .Net Framework 3.5 SP1
ArcGIS Uninstall Utility	Read Me	1.23 MB	Download English
ArcGIS for Desktop	Install Guide	822.63 MB	Download English
ArcGIS Data Interoperability for Desktop		452.70 MB	Download English
ArcGIS Data Reviewer for Desktop	Install Guide	84.28 MB	Download English
ArcGIS Workflow Manager for Desktop	Install Guide	55.32 MB	Download English
ArcGIS Tutorial Data for Desktop		1.65 GB	Download English

Figura A.1 – Selección del producto ArcGis a utilizar

NOTA: Se recomienda utilizar el gestor de descargas que proporcionan, ya que el programa ocupa alrededor de un 1 GB y la descarga regular suele quedarse estancada.

Seguimos las instrucciones que se proporcionan para la instalación y la activación.

Una vez hemos completado el paso anterior, en el menú **Inicio** > **Todos los programas** > **ArcGis** encontraremos la suite de programas. Utilizaremos en este caso la herramienta **ArcMap** para poder visualizar los **ShapeFiles** sobre un mapa.

➲ Cambio de coordenadas en los ShapeFiles usando ArcMap

En este apartado explicaremos como transformar las coordenadas de un dataset en formato **ShapeFile**. Los datasets proporcionados por el Ayuntamiento de Madrid vienen dados en el sistema de coordenadas **ED50**, mientras que los mapas que utilizamos para mostrar los recursos (Google Maps) utiliza el sistema **WGS84,** por lo que es necesaria esta transformación para que no aparezcan desviados los recursos sobre el mapa de Google Maps. Para este ejemplo, vamos a transformar las coordenadas del dataset **Aparcabicis.**

➲ Establecer un mapa base

Comenzamos con un proyecto en blanco. A continuación, hacemos clic en el menú **Archivo > ArcGis Online** e iniciamos sesión si aún no lo hemos hecho.

En la ventana que aparece podemos buscar mapas ya publicados por otros usuarios en ArcGis Online. En el cuadro ***Buscar***, escribimos **openstreetmap** y seleccionamos cualquiera de los 3 que aparezca cuyo publicador sea **esri**. Pulsamos en ***Abrir*** y el mapa se importará al programa.

OpenStreetMap es un mapa gratuito y de código abierto que publica ESRI de manera oficial para comenzar a trabajar. No son los mismos mapas que utiliza Google Maps, sin embargo utiliza el mismo sistema de coordenadas: el WGS84. De esta manera luego intercambiaremos el mapa base por el de Google Maps y el resultado será el mismo.

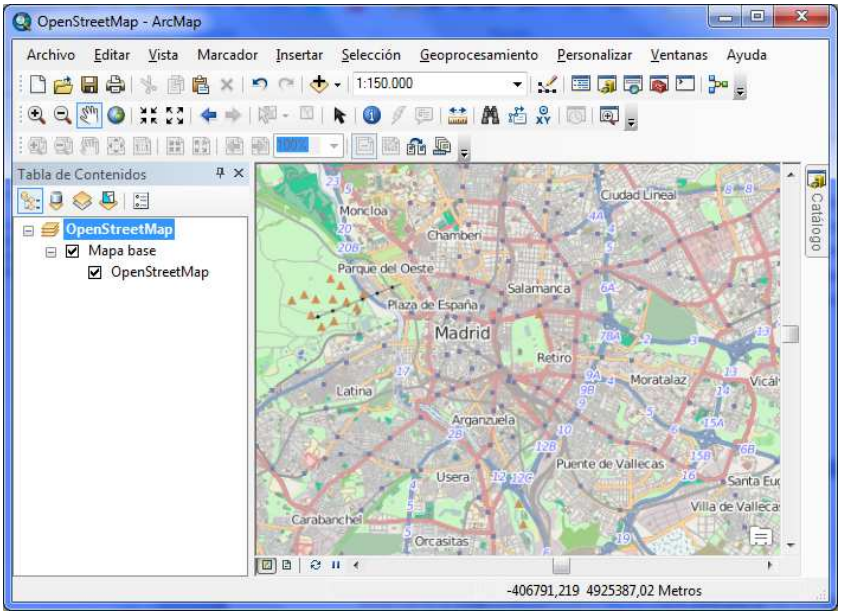


Figura A.2 – ArcMap con OpenStreetMap como mapa base

➲ Insertar un ShapeFile (SHP)

En el lateral derecho del programa hay un botón llamado Catálogo. Al pulsar sobre él se despliega una vista en la que podemos ver diferentes fuentes de recursos. Nosotros nos centraremos en **Conexiones a carpetas**. Esto es, que carpetas del sistema de ficheros de nuestra maquina queremos que sean accesibles desde ArcMap.

Una vez tengamos acceso a la carpeta que contiene nuestro proyecto ShapeFile, seleccionamos el archivo ***130111_Aparcibicis.shp*** y lo arrastramos a la Tabla de contenidos (panel del lateral izquierdo).

Como el ShapeFile está en un sistema de coordenadas (ED50) diferente al de *openstreetmap* (WGS84), aparecerá un diálogo notificando el problema. Pulsamos en el botón ***Transformaciones…*** y ajustamos las opciones del nuevo dialogo que aparece de la la figura A.3

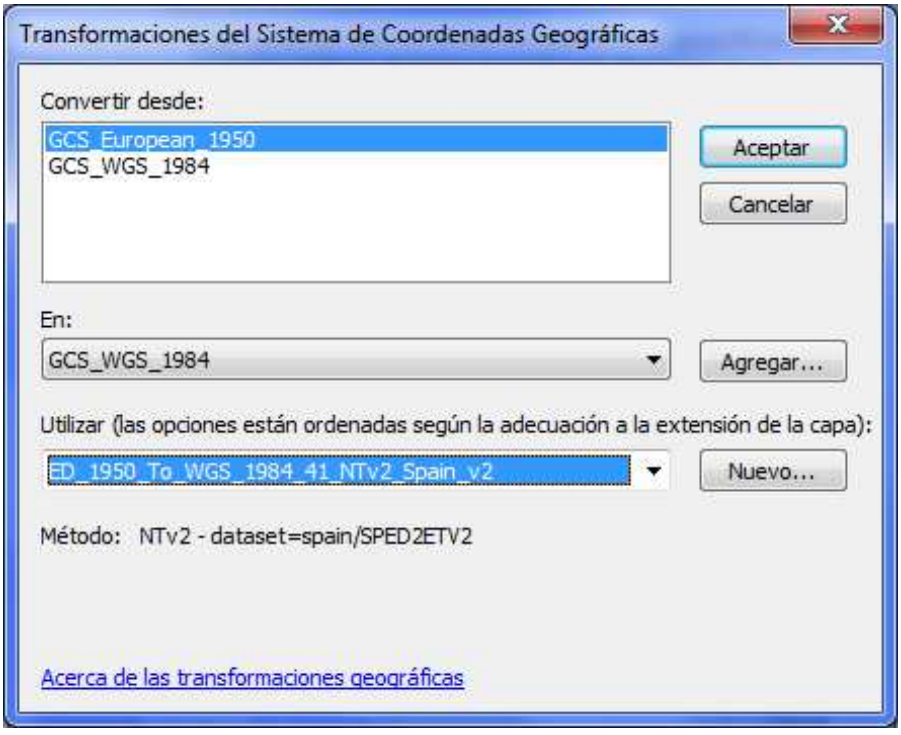


Figura A.3 – Transformaciones del Sistema de Coordenadas

Es importante utilizar el método NTv2 para una correcta visualización. Una vez terminado, podemos comprobar que ahora los marcadores del mapa están bien ajustados. Los 4 aparcabicis situados a las puertas del Jardín Botánico de la Complutense aparecen correctamente situados.

Figura A.4 – AparcaBicis marcados con puntos azules a las puertas del Jardín Botánico de la Complutense

El último paso que nos queda, sería hacer definitivas estas transformaciones en el ShapeFile, ya que con esto solo hemos conseguido visualizarlas bien en el mapa de ArcMap.

➲ Transformar las coordenadas de un ShapeFile (SHP)

Pulsamos sobre el menú **Geoposicionamiento** > **ArcTool Box** > **Herramientas de administración de datos** > **Proyecciones y transformaciones** > **Proyectar** y configuramos las opciones de la siguiente manera:

- Como **entidad de entrad**a seleccionamos el archivo *.shp* que queremos transformar, en nuestro caso el ShapeFile de Aparcabicis.

- En **Sistema de coordenadas de salida**, en el formulario que aparece hacemos clic en la carpeta Capas (abajo del todo), que ya contiene la configuración actual del OpenStreetMap (figura A.5)

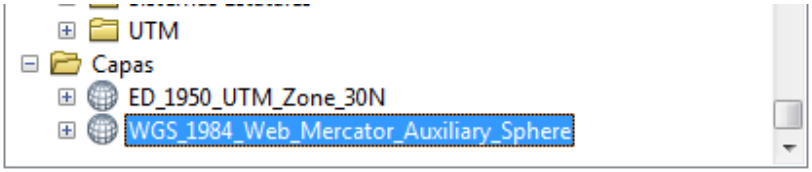


Figura A.5 – Sistema de coordenadas del OpenStreetMap

El formulario debería tener un aspecto parecido al de la figura A.6 salvo por las rutas de los ficheros:

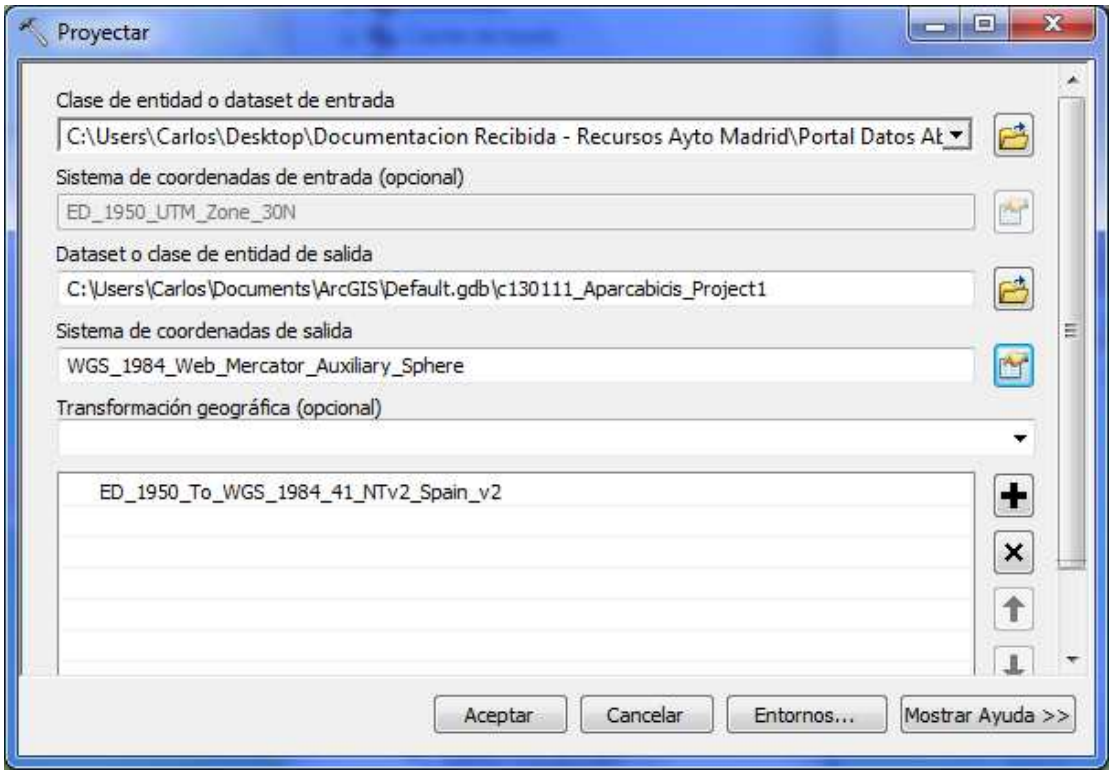


Figura A.6 – Configuración de la herramienta Proyectar

Una vez configurados, pulsamos en **Aceptar** y le dejamos trabajar unos minutos. Veremos en la barra de estado de ArcMap que se muestra un banner con la palabra ***Proyectando...***. Cuando finalice, ArcMap lo notificará y tendremos una nueva capa con las transformaciones de coordenadas hechas. Esta copia aparece en la Tabla de contenidos.

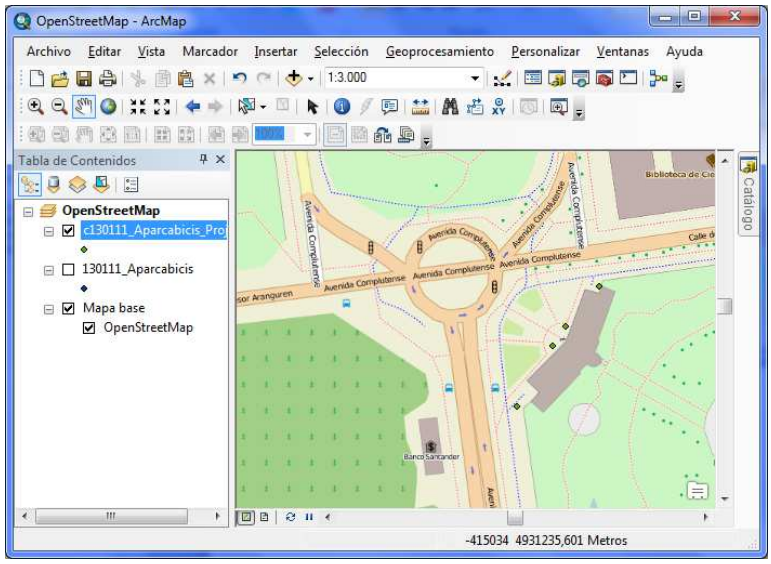


Figura A.7 – Nueva capa con las proyecciones en el Sistema de Coordenadas correcto

Para comprobar que se ha realizado correctamente, hacemos clic derecho sobre esta capa nueva y en **Propiedades** > **Fuente** comprobamos que el sistema de coordenadas proyectadas ya está en WGS84.

➲ Exportar los cambios

Pulsamos de nuevo sobre esta nueva capa con el botón de derecho > Datos > Exportar datos… y configuramos el formulario de la siguiente manera:

- Exportamos **Todas las entidades.**
- Utilizamos el mismo sistema de coordenadas que **datos de fuente de capa.**
- **Clase de entidad de salida** pulsamos sobre el botón y seleccionamos donde queremos guardar el SHP exportado. Le damos un nombre al archivo y seleccionamos el Tipo de archivo a exportar en *Shapefile*.

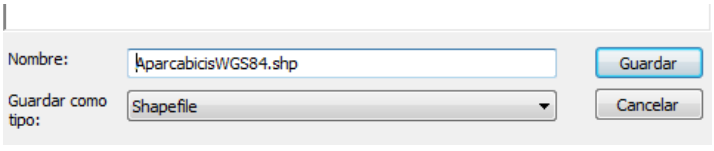


Figura A.8 – Guardar fichero SHP con las transformaciones realizadas

Anexo II – Transformar SHP en CSV

Existen dos formas de transformar un fichero SHP a CSV, que dependen de la disponibilidad de los sitios web que se usan en ellos.

❋ Método A:

Este método utiliza el sitio web SHP Escape (http://shpescape.com). Dado que es un sitio web experimental, puede ocurrir que en ocasiones, el servidor no esté disponible. Si tenemos la suerte de que se encuentra en funcionamiento, tenemos que:

- Accedemos a la página y seleccionar la opción de convertir SHP a Tabla Fusión de Google.
- A continuación cargamos el fichero ZIP (sin subcarpetas) que contiene todos los ficheros del proyecto ShapeFile y pulsamos en **Next**.
- Esperamos un poco, ya que la solicitud tarda en llevarse a cabo. Una vez que finalice, automáticamente nos mostrará la tabla dinámica generada. Si queremos recuperarla más tarde, esta tabla queda guardada en el almacenamiento virtual de Google Drive.
- Una vez tengamos la tabla, pulsamos en el menú ***File…*** > ***Download*** y en el cuadro de diálogo que aparece, seleccionamos el formato CSV, para descargar la copia en dicho formato.

❋ Método B:

Si el sitio web del método anterior falla, tenemos como alternativa utilizar otro sitio web a cambio de realizar algunos pasos adicionales. En esta caso, utilizaremos un convertidor de ArcGis (que también se encuentra en fase experimental) que convierte de SHP a KML (y a otros formatos). Después podremos usar este KML para importarlo en las tablas dinámicas:

- Accedemos a la página http://converter.mygeodata.eu/ y seleccionamos la opción **Vector**
- A continuación cargamos el fichero ZIP (sin subcarpetas) que contiene todos los ficheros del proyecto SHP.
- En la pantalla siguiente aparecen los resultados del análisis del fichero cargado. Aquí podemos comprobar que los datos sean correctos una vez más. Pulsamos

sobre el botón ***Check available operations*** que se encuentra en la parte inferior de la página

- Entre las opciones disponibles, tan solo debemos configurar la opción ***Export to format*** seleccionando KML. El resto de opciones se dejan por defecto. Para terminar pulsamos sobre ***Proceed selected operations***

- Una vez concluido, nos mostrará un link para descargar un ZIP en cuyo interior se encuentra el fichero KML. Lo descargamos y descomprimimos.

- Ahora, abrimos Google Drive y pulsamos en ***Crear*** > ***Tabla Dinámica***. Si la opción no está disponible, pulsamos en ***Conectar más aplicaciones*** y en la ventana que aparece, buscamos ***"tablas dinamicas".*** Conectamos con la aplicación experimental de Tablas Dinámicas de Google.

- En el asistente de creación de tablas, seleccionamos ***Import new table... From this computer*** y examinamos el equipo en busca del KML descomprimido del ZIP descargado anteriormente. Pulsamos en ***Next.***

- El resto de opciones se deja por defecto, por lo que pulsamos ***Next*** hasta el final del asistente donde pulsaremos en ***Finish***

- Una vez tengamos la tabla importada, pulsamos en el menú ***File...*** > ***Download*** y en el cuadro de diálogo que aparece, seleccionamos el formato CSV, para descargar la copia en dicho formato.

❋ Pasos comunes a ambos métodos:

Con cualquiera de estos dos métodos obtenemos el dataset en formato CSV, aunque aún falta un paso adicional para mostrar las coordenadas geográficas sin código KML. Este paso solo se aplica para aquellos datasets que tratan marcadores sobre mapas, puesto que se trata de un único punto, es decir, un solo par de coordenadas. Sin embargo, para los datasets de vías ciclistas que pintan segmentos de línea sobre los mapas en lugar de puntos, no es aplicable, debido a que no se conoce a priori el número de segmentos (número de pares de coordenadas) a pintar.

Abrimos el CSV obtenido con un editor de texto avanzado y usando ***Find & Replace*** cambiamos lo siguiente:

Buscar	Sustituir por...	Veces
geometry	LATITUD, LONGITUD	Solo la primera coincidencia
“<Point>		Sustituir todos
<coordinates>		
</coordinates>		
</Point>”		

El resultado final debe parecerse a:

Campo: geometry

Valor: <Point><coordinates>”40.32423,-3.7454”</coordinates></Point>

Por:

Campos: LATITUD, LONGITUD

Valor: 40.32423,-3.7454

Es decir, cambiamos la columna **geometry** por dos: **LATITUD** y **LONGITUD**. También eliminamos las etiquetas KML (<Point> y <coordinates>). Además, eliminamos las comillas dobles para separar las coordenadas en las nuevas columnas creadas; si no se borran, aparecerán ambas coordenadas en la misma columna